



01

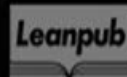
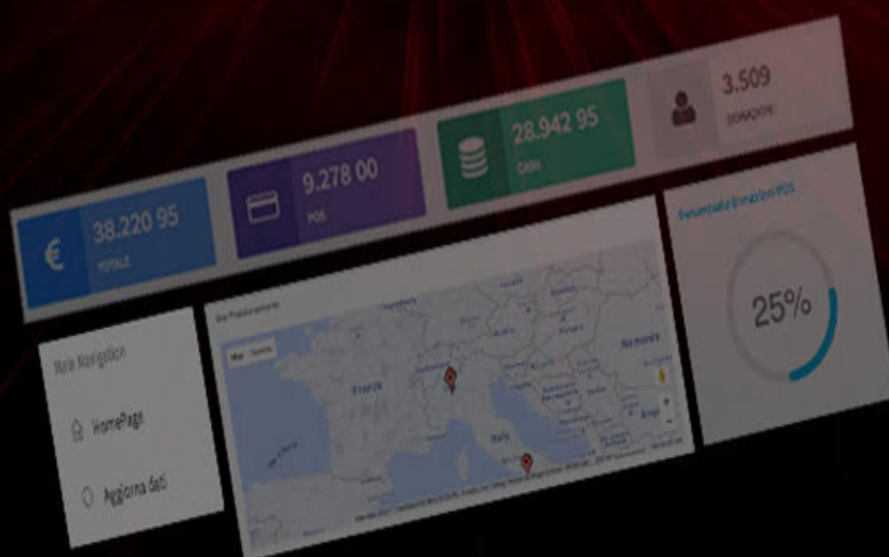
Come Sviluppare un'applicazione Ruby on Rails
Teoria ed Esercizi Pratici

Flavio Bordoni

Roberto De Santis

Donamat Dashboard

“Un cammino passo a passo nella creazione di un'applicazione professionale con l'uso del potente framework Ruby on Rails applicato a concetti di 'Standard Web Design'.”



Donamat Dashboard

Flavio Bordoni e Roberto De Santis

Questo libro è in vendita presso <http://leanpub.com/donamatonrails>

Questa versione è stata pubblicata il 2019-01-26



Questo è un libro di [Leanpub](#). Leanpub permette ad autori ed editori un processo di pubblicazione agile. La [Pubblicazione Agile](#) consiste nel pubblicare un ebook in corso d'opera, utilizzando strumenti leggeri e molte iterazioni per ottenere un feedback dai lettori, al fine di assicurare un libro giusto e attraente una volta completato.

© 2016 - 2019 Flavio Bordoni e Roberto De Santis

Indice

Riconoscimenti	i
Prefazione	ii
Requisiti per chi legge il libro	iii

Definiamo l'ambiente di sviluppo. 1

Nuova applicazione Rails 5 utilizzando il workspace di Cloud9.	2
Che cos'è Cloud9	2
Nuovo workspace su Cloud9	2
Installiamo Rails 5	4
Creiamo l'applicazione	4
Apriamo l'applicazione localmente	4

Git	6
Implementiamo Git	6
inizializziamo	6
Verifichiamo configurazione globale	7
Escludiamo files da git	7
Versione iniziale v0.1.0	8
Branch	8
Creiamo un nuovo ramo e ci spostiamo su esso.	8
Lavoriamo sul codice e aggiorniamo il ramo	8
Chiudiamo il ramo	9
Considerazioni finali del capitolo	9

PostgreSQL	10
Apriamo il branch	10
Implementiamo PostgreSQL	10
Chiudiamo il branch	11

Settaggi Base	13
Apriamo il branch	13
Fissiamo la versione di Ruby sul Gemfile	13
Chiudiamo il branch	14

Pagina di test	15
Apriamo il branch	15
No scaffold	15
Instradiamo con routes	16
Creiamo testpage views	16
Chiudiamo il branch	16
Figaro	18
Apriamo il branch	18
Perché usare figaro?	18
Installiamo figaro	19
Cominciamo a mettere al sicuro le password	21
Giochiamo con le variabili d'ambiente (ENV)	23
Chiudiamo il branch	23
Heroku	25
Apriamo il branch	25
Heroku piattaforma cloud PaaS	25
Che cosa è una piattaforma cloud PaaS	25
Che cosa è Heroku	25
La storia di Heroku	26
A cosa serve e come funziona Heroku	26
Heroku come repository remoto	27
Inizializziamo Heroku	27
Settaggio per la pubblicazione su Heroku	29
Creiamo l'app su heroku	29
Chiudiamo il branch	30
Internazionalizzazione della pagina di test.	32
Apriamo il branch	32
Internazionalizzazione (i18n)	32
I18n statico con YAML	32
Scelgo lingua di default	34
Chiudiamo il branch	34
Github	36
Apriamo il branch	36
L'importanza del README.md	36
Github	37
Chiudiamo il branch	42

Riconoscimenti

prima edizione

Flavio

dedico questo libro a mia moglie Elisa, che mi ha sposato, il più grande colpo di fortuna della mia vita, Ai miei fratelli e sorella Angelo, Valerio e Alice per contribuire quotidianamente ad alimentare la mia passione per la programmazione e il web. A mia Madre che mi appoggia su ogni cosa che mi viene in mente, anche quelle sbagliate. La mia vera musa ispiratrice. A mio Padre che ha sempre voluto che scrivessi un libro.

Roberto

dedico questo libro a mia moglie Daisy e ai miei figli Valerio e Alessandro, che sono fonti di ispirazione quotidiana. Ringrazio mio padre Alberto e mio fratello Andrea per avermi spinto a scrivere questo libro.

Flavio & Roberto

ringraziano Giovanni Cavallo, Giacomo Tirillò e Giuseppe Castana di Itineris Italia perchè senza di loro non esisterebbe il progetto 'Donachiaro' e di conseguenza questo libro.

Inoltre ringraziamo altre due fonti che ci hanno influenzato e motivato come Michael Hartl con il suo Tutorial su Ruby On Rails e Jeffrey Zeldman con la sua 'bibbia' sul Web Standard Design.

Prefazione



Dr. Giovanni Cavallo

di Giovanni Cavallo - Sales Manager & Co-Founder Itineris Italia

DONACHIARO è il servizio di Itineris Italia dedicato alle Organizzazioni Non Profit e agli Enti Pubblici e Privati che promuovono campagne di raccolta fondi a scopo sociale. L'obiettivo del servizio è consentire una gestione trasparente della raccolta fondi, garantendo chiarezza dell'informazione, semplificazione dei processi di donazione, tracciabilità dei flussi, snellimento delle procedure fiscali. L'area dedicata ai nostri clienti è stata pensata secondo criteri di trasparenza e semplicità di gestione, elementi fondamentali per ogni attività di fundraising. Ogni organizzazione che ha scelto il servizio Donachiaro dovrebbe poter accedere con le proprie User ID e password alla propria dashboard personalizzata, dove viene riportato in tempo reale il dettaglio delle donazioni per ogni stazione utilizzata. Nella dashboard dovranno essere disponibili i dati delle donazioni effettuate in contanti e con moneta elettronica, con la descrizione delle singole transazioni (ora, importo, sistema di versamento) e, ove inseriti, i dati anagrafici del sostenitore autorizzati al trattamento. I dati di raccolta potranno essere ordinati e analizzati secondo diversi criteri e possono essere oggetto di grafici e statistiche, confrontati rispetto a determinati periodi di tempo ed analizzati secondo le variabili presenti nei periodi di fundraising. Il sistema elaborerà i valori di donazione media per periodo e per sistema di donazione e creerà in automatico un database dei donatori che lasciano, previa autorizzazione al trattamento dei dati, le informazioni anagrafiche sulla stazione Donachiaro. Questo libro narra di come la ns. visione si è trasformata in una reale applicazione fruibile dai nostri clienti.

Giovanni Cavallo

Requisiti per chi legge il libro

Per leggere il libro è necessario avere:

- Conoscenza base del framework Ruby on Rails
- Concetti base di HTML
- Concetti base di CSS

Definiamo l'ambiente di sviluppo.

Nuova applicazione Rails 5 utilizzando il workspace di Cloud9.

Che cos'è Cloud9

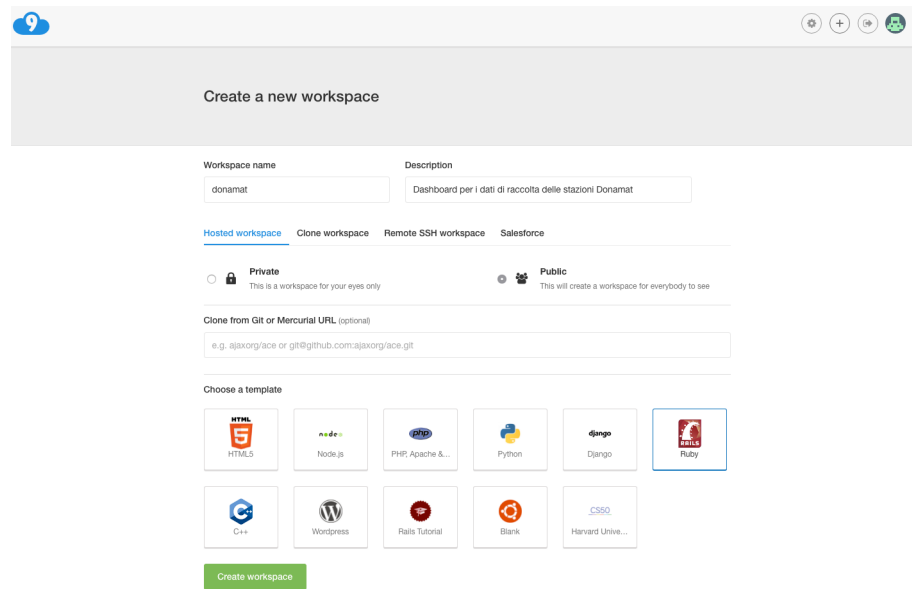
Cloud9 è un ambiente di sviluppo sul cloud. Ruby on rails è standardizzato per l'ambiente di sviluppo cloud9. L'ambiente di lavoro cloud9 si presenta pre-configurato con la maggior parte del software necessario per lo sviluppo professionale in Rails e include Ruby, RubyGems e Git. La Cloud IDE (integrated development environment) include anche tre componenti essenziali per sviluppare applicazioni web: un editor di testo, un navigatore del filesystem e un terminale per la riga di comando. Cloud9 consente agli sviluppatori di collaborare tra loro in team con caratteristiche che amplificano il supporto di ogni sviluppatore, anche real time. Inoltre garantisce la possibilità di verificare l'andamento dello sviluppo con anteprime in tempo reale e test di compatibilità dei browser. Cloud9 usa per i suoi workspaces Docker ed è osteggiato su Google Compute Engine. Docker è un progetto open-source che automatizza lo sviluppo di applicazioni all'interno di contenitori software. I contenitori di Docker hanno parti di software in un completo filesystem che prevedono l'uso di tutto ciò che occorre per lo sviluppo e il deployment di una web application: codice, runtime, tools di sistema, librerie di sistema. Docker fornisce un addizionale layer astratto dall'hardware del sistema operativo virtualizzato su Linux.

Nuovo workspace su Cloud9

Su cloud9 non partiamo dal workspace con template Rails precaricato perché c'è una versione rails più vecchia e non ha postgresQL preinstallato. Quindi iniziamo con un workspace "blank" ed installiamo tutto rails. Iniziamo con il creare un nuovo workspace su cloud9 dal titolo: donamat

Nuova applicazione Rails 5 utilizzando il workspace di Cloud9.

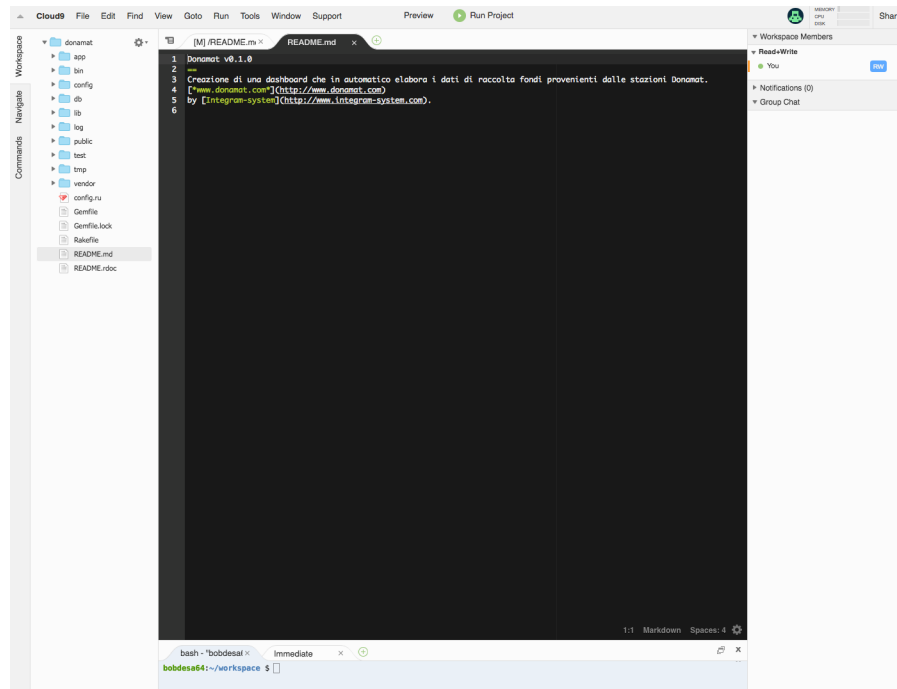
3



The screenshot shows the 'Create a new workspace' interface in Cloud9. At the top, there's a header with the Cloud9 logo and navigation icons. The main heading is 'Create a new workspace'. Below this, there are two input fields: 'Workspace name' with the value 'donamat' and 'Description' with the value 'Dashboard per i dati di raccolta delle stazioni Donamat'. Underneath, there are tabs for 'Hosted workspace', 'Clone workspace', 'Remote SSH workspace', and 'Salesforce'. The 'Hosted workspace' tab is selected. Below the tabs, there are two radio buttons for 'Private' (selected) and 'Public'. A text field for 'Clone from Git or Mercurial URL (optional)' contains the example URL 'e.g. ajaxorg/ace or git@github.com:ajaxorg/ace.git'. A section titled 'Choose a template' displays a grid of icons for various technologies: HTML5, Node.js, PHP, Apache &..., Python, Django, Ruby (selected), C++, Wordpress, Rails Tutorial, Blank, and CSSO. At the bottom, there is a green 'Create workspace' button.

creazione workspace c9

A questo punto non resta che inserire le informazioni che descrivono il progetto anche nel file readme.md



c9 readme

Installiamo Rails 5

In produzione Heroku utilizza PostgreSQL quindi lo installiamo anche localmente. Possiamo gestire PostgreSQL localmente nell'ambiente di sviluppo e test perché su cloud9 è già preinstallato PostgreSQL e dobbiamo solo farlo partire. Un'alternativa era quella di caricare la gemma “pg” solo per l'ambiente di produzione. Ma se possibile è preferibile usare nell'ambiente di sviluppo le stesse risorse usate in produzione.

terminal

```
$ gem install rails -v 5.0.0.1
```

Creiamo l'applicazione

Una volta installato Rails passiamo a creare l'applicazione

terminal

```
$ cd ~/workspace  
$ rails _5.0.0.1_ new donamat --database=postgresql
```

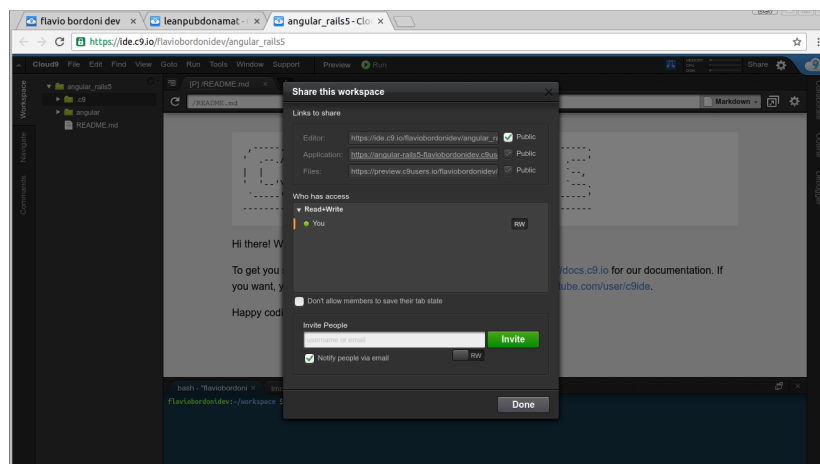
Apriamo l'applicazione localmente

Per aprire la nuova applicazione entriamo nella cartella elisinfo5 e facciamo partire il web server

terminal

```
$ cd donamat  
$ rails s -b $IP -p $PORT
```

su cloud9 andiamo sul link “Share” in alto a destra e clicchiamo sul link di “application” per aprire l'applicazione.



application: link



se stiamo lavorando su un workspace “shared” non avremo il link “Share” in alto a destra. Per capire quale è l’URL da mettere sul browser dobbiamo fare ... (già scritta questa parte cerca negli altri libri)

Abbiamo un errore causato da postgresQL che non è attivo sul nostro ambiente.

lo risolviamo nel prossimo capitolo perché la prima cosa da fare dopo la creazione di una nuova applicazione è implementare Git.

Git

Version Control System. Un gestore che controlla le versioni del software in fase di sviluppo.

Implementiamo Git

implementare git richiede i pochi comandi presentati su questo paragrafo. Se volete implementarlo con più calma usando anche altri comandi di controllo passate al paragrafo successivo.

Sommario del capitolo Git

```
$ cd donamat
$ git init

$ git add -A
$ git commit -m "new rails app"

$ git tag v0.1.0
```



Ricordiamoci di entrare nella directory “donamat” altrimenti si inizializza git per tutto il workspace e questo è un problema per quando si fa il push su heroku.

inizializziamo

Vediamo che non ci sono le cartelle nascoste di git

terminal

```
$ ls -a
```

inizializziamo git.

terminal

```
$ git init
```

Questo genera anche la cartella nascosta git ed i vari files tra cui gitignore per vedere possiamo visualizzare i files nascosti dalla ruota dentata al lato del nome del workspace scegliendo dal menu a discesa la voce “show hidden files” Oppure da terminale

terminal

```
$ ls -a
```

Verifichiamo configurazione globale

Verifichiamo la configurazione globale di git che è utilizzata per i repository esterni quali Github, Heroku, Gitbuket, ... Per verificare le impostazioni di git eseguiamo

terminal

```
$ git config -l
```

Se serve modificarle

terminal

```
$ git config --global user.name "Your Name"
$ git config --global user.email "your@email.com"
$ git config --global push.default matching
```

Escludiamo files da git

Escludiamo da git quei files che non fanno parte del codice. Ad esempio i files temporanei o quelli di log o file di configurazione di pacchetti accessori [Gitignore - ignoring files](https://help.github.com/articles/ignoring-files)¹. Altri files che è importante escludere sono quelli riguardanti password/secrets. Ma questi ultimi li trattiamo nel prossimo capitolo che si intitola Figaro.

Cloud9 nel workspace rails inserisce un gitignore di default che al momento lasciamo così ma che modificheremo in seguito

.gitignore

```
1 # See https://help.github.com/articles/ignoring-files for more about ignoring files.
2 #
3 # If you find yourself ignoring temporary files generated by your text editor
4 # or operating system, you probably want to add a global ignore instead:
5 #   git config --global core.excludesfile '~/.gitignore_global'
6
7 # Ignore bundler config.
8 /.bundle
9
10 # Ignore the default SQLite database.
11 /db/*.sqlite3
12 /db/*.sqlite3-journal
13
14 # Ignore all logfiles and tempfiles.
15 /log/*
16 !/log/.keep
17 /tmp
```

¹<https://help.github.com/articles/ignoring-files>

Versione iniziale v0.1.0

Effettuiamo il primo commit ed eseguiamo anche il tag v.0.1.0 in accordo con la convenzione del [semantic versioning](http://semver.org)²

terminal

```
$ git add -A
$ git commit -m "new rails app"

$ git tag v0.1.0
```

il tag ci permette di distinguere delle “pietre miliari” durante lo sviluppo. Tappe che vengono identificate da un cambio del numero della versione. Il numero di versione segue le convenzioni del semantic versioning (semver.org)

Branch

Il nome del ramo principale in Git è master. Quando si inizia a fare dei commit, li stai dando al ramo master che punterà all’ultimo commit che hai eseguito. Creando un nuovo ramo o branch il puntatore si sposterà su quest’ultimo lasciando inalterato il ramo master.

Nei prossimi capitoli utilizzeremo SEMPRE il Branch per fare le modifiche e poi lo eliminiamo dopo aver passato le modifiche sul master (operazione di merge). Ogni volta che facciamo una modifica “anche piccola” apriamo un branch. Il branch ci permette di tornare indietro se abbiamo fatto casini. Fare un branch anche per piccole modifiche ci limita problemi di “merge” da risolvere.

Creiamo un nuovo ramo e ci spostiamo su esso.

terminale

```
$ git checkout -b nomebranch
```



Non è indispensabile creare e spostarsi sul nuovo branch prima di fare le modifiche perché le modifiche vengono passate su git solo quando si fa il `** git add **`. E’ comunque buona prassi iniziare creando il branch prima di modificare il codice.

Lavoriamo sul codice e aggiorniamo il ramo

²<http://semver.org>

terminale

```
$ git add -A
$ git commit -m "modifiche fatte"
```

```
$ git add -A
$ git commit -m "modifiche fatte2"
```

```
$ git add -A
$ git commit -m "modifiche fatte3"
```

```
...
```

Chiudiamo il ramo

se abbiamo finito le modifiche e va tutto bene:

Riassunto del capitolo

```
$ git checkout master
$ git merge nomebranch
$ git branch -d nomebranch
```

se abbiamo fatto casino e vogliamo tornare indietro abortando il branch:

Riassunto del capitolo

```
$ git checkout master
$ git branch -D nomebranch
```

Considerazioni finali del capitolo

Il version control system Git è utile per:

1. tener traccia dell'andamento dei lavori.
2. avere tranquillità di poter tornare indietro con i branches
3. pubblicare la nostra applicazione su Heroku

PostgreSQL

Risolviamo l'errore di connessione creando i databases

Apriamo il branch

terminal

```
$ git checkout -b pg
```

Implementiamo PostgreSQL

nel primo capitolo abbiamo creato una nuova applicazione usando i settaggi del database postgresQL ** \$ rails 5.0.0.1 new my_app_name --database=postgresql ** adesso implementiamo postgresQL sul nostro workspace di cloud9.

PostgreSQL è preinstallato su ogni workspace di Cloud9, basta attivarlo.



Il “sudo sudo” su alcuni comandi non è un errore di digitazione ma è necessario su Cloud9 per evitare che il prompt ti richieda una password per l'utente ubuntu interrompendo il comando perché non viene fornita una password per l'utente ubuntu.

terminal

```
$ sudo service postgresql start
```

Verifichiamo sul web

terminal

```
$ cd donamat  
$ rails s -b $IP -p $PORT
```

Vediamo che l'errore è cambiato. Adesso si connette a PostgreSQL ma non trova i databases di sviluppo e di test. Questi sono definiti sul file /config/database.yml

Come si vede sul database.yml di postgresql diamo il nome dei database con la convenzione “nome-applicazione + _development (o _test)” Non creiamo il database di produzione (_production) perché la produzione la teniamo su heroku.

Adesso non ci resta che creare il database sul postgresQL del workspace di cloud9. Il servizio postgresql è già attivo quindi creiamo i databases

terminal

```
$ createdb my_app_name_development
$ createdb my_app_name_test
```

Per verificare i databases dalla linea di comando di PostgreSQL:

terminal

```
$ psql
postgres=# \list
postgres=# \q
```

Avremmo potuto creare i databases anche da dentro la linea di comando di postgresQL:

terminal

```
$ psql
postgres=# CREATE DATABASE "my_app_name_development";
postgres=# CREATE DATABASE "my_app_name_test";
postgres=# \list
postgres=# \q
```

**ATTENZIONE**

il database ha encoding: SQL_ASCII quindi non supporta caratteri accentato come invece fa UTF8
Probabilmente è bene cambiare encoding...

verifichiamo che c'è comunicazione eseguendo

terminal

```
$ rake db:migrate
```

Va a buon fine quindi non è necessario impostare un utente ed una password per il database. Quelle impostate in automatico di default vanno bene.

terminal

```
$ git add -A
$ git commit -m "implement postgresQL"
```

Chiudiamo il branch

se abbiamo finito le modifiche e va tutto bene:

terminal

```
$ git checkout master  
$ git merge pg  
$ git branch -d pg
```

Settaggi Base

Apriamo il branch

terminal

```
$ git checkout -b basic
```

Fissiamo la versione di Ruby sul Gemfile

Scriviamo sul gemfile con che versione di ruby è stata create così tutte le volte che farà un bundle avrà un allarme se è su un ambiente rbenv con la versione di ruby errata.

verifichiamo la versione di ruby che stiamo usando

terminal

```
$ ruby -v
```

e la scriviamo sul gemfile

Gemfile

```
3 # versione di ruby usata
4 ruby '2.3.0'
```



La versione va riportata senza il numero di patch ('2.3.0' e non '2.3.0p0'). Se si riporta anche la patch ad esempio '2.3.0p0' si avrà un messaggio di errore nel momento in cui faremo il "bundle install".

Ed una volta usato il "bundle install" l'installato viene registrato sul file: Gemfile.lock

terminal

```
$ bundle install

$ bundle update

$ bundle install
```



Lo stesso Heroku che tratteremo più avanti da un warning se non trova la versione di ruby fissata

terminal

```
$ git add -A  
$ git commit -m "add ruby version"
```

Chiudiamo il branch

se abbiamo finito le modifiche e va tutto bene:

terminale

```
$ git checkout master  
$ git merge basic  
$ git branch -d basic
```

Pagina di test

Pagina di test dello sviluppatore. Inizio con delle testpages di appoggio per lo sviluppatore. Così da poter implementare l'internazionalizzazione ed il login sul back-office

Apriamo il branch

terminal

```
$ git checkout -b testpages
```

No scaffold

non usiamo lo Scaffold perché la homepage non ha una corrispettiva tabella nel database ma prende i dati da altre tabelle. Uso quindi il “rails generate controller ...” e gli associo solo l’azione “index”.

Perché “index” che sarebbe un elenco e non “show” che è la pagina che presenta i dati?

Non ho una risposta definitiva ed in passato ho usato “show” ma ultimamente sto usando index perché è più comunemente usato in html e perché se penso all'url con index non ho la necessità di specificare l'id come faccio invece su show. (es. mydomain/homepages invece di mydomain/homepages/1) A questo punto evito anche di usare il plurale come fa lo “scaffold” perché non ho una tabella “homepages”. Uso quindi il singolare visto che è un controller suigeneris e usa l’“index” come se fosse “show” perché non ha un elenco di elementi. Ho quindi mydomain/homepage.



ATTENZIONE: con “rails generate controller ...” -> uso il SINGOLARE (ed ottengo un controller al singolare)

Poiché è una pagina statica lascio il controller al singolare visto che non ha un elenco di elementi da visualizzare

terminal

```
$ rails g controller Testpages page_a page_b
```

Creiamo due azioni page_a page_b per avere due pagine per fare delle prove con un link che va da una pagina all'altra.

non ho nessun migrate perché non mi interfaccio con il database.

Vediamo la nostra applicazione rails funzionante:

terminal

```
$ cd donamat
$ rails s -b $IP -p $PORT
```

<https://donamat-flaviobordonidev.c9users.io/> https://donamat-flaviobordonidev.c9users.io/homepage/page_a - https://donamat-flaviobordonidev.c9users.io/homepage/page_b

Instradiamo con routes

Aggiorno il file routes per mettere l'homepage come pagina principale (root)

config/routes.rb

```
3 root 'testpages#page_a'
4 get 'testpages/page_a'
5 get 'testpages/page_b'
```

Creiamo testpage views

views/testpage/index.html.erb

```
1 <h1> Questa è il testpage </h1>
2 <p> Il testo verrà preso dal database ma alcuni messaggi sono passati dall'applicazione ed è quindi bene che vengano trad\
3 otti per essere pronti a supportare più lingue. </p>
4
5 <br>
6 <p> <%= link_to 'andiamo al test', homepage_testme_path %> </p>
```

views/testpage/testme.html.erb

```
1 <h1> Benvenuti nella pagina di test </h1>
2 <p> <%= link_to 'Torniamo in testpage index', testpage_index_path %> </p>
3 <p> <%= link_to 'Torniamo in testpage', root_path %> </p>
```

Posso verificare che funziona tutto sull'url:

Chiudiamo il branch

se abbiamo finito le modifiche e va tutto bene:

terminal

```
$ git checkout master  
$ git merge testpage  
$ git branch -d testpage
```

Figaro

Un problema di sicurezza. Le password e le chiavi di criptatura non sono state escluse da git e vengono quindi passate sui repositories esterni. Questo non è cosa buona e giusta. Implementiamo la sicurezza con la gemma Figaro.

Apriamo il branch

terminal

```
$ git checkout -b figaro
```

Perché usare figaro?

Figaro è una gemma vecchiotta che però anche per Rails 5 fa egregiamente il suo lavoro. Da rails 4.1 è stato introdotto il file secrets.yml che è un'alternativa a figaro. Però figaro vale ancora la pena...

Rails 4.1 ha introdotto il file secrets.yml che potrebbe sostituire figaro ma non al 100%. Il problema è che secrets.yml non setta le variabili d'ambiente (environment variables). Lo stesso secrets.yml di default ne usa una per l'ambiente di produzione.

config/secrets.yml

```
47 # Do not keep production secrets in the repository,  
48 # instead read values from the environment.  
49 production:  
50   secret_key_base: <%= ENV["SECRET_KEY_BASE"] %>
```

Quindi dove raccolgo tutte le password per la mia applicazione per poi passarle come variabili d'ambiente? Al momento conviene ancora usare figaro e raccoglierle sul file a lui dedicato config/application.yml All'avvio di rails la gemma figaro carica in variabili d'ambiente tutte le password scritte su config/application.yml

Evita di usare .bashrc o .bash-profile per le passwords/secrets

Quando immagazzini passwords/secrets in file come .bashrc, queste sono mandate come variabili d'ambiente ad ogni singolo programma che stai eseguendo come utente. La maggior parte di questi programmi non ha bisogno di conoscere le tue passwords/secrets. Quindi perché passarglieli? Rivela le tue passwords/secrets solo ai processi che ne hanno bisogno. Gemme come figaro o detenv ti permettono di aggiungere variabili d'ambiente ai tuoi files che sono caricati all'avvio di Rails. Quest variabili d'ambiente saranno disponibili solo al processo Rails e ai suoi processi figli.

**ATTENZIONE!**

ricordiamoci di mettere config/application.yml su .gitignore per non passare il file delle password su git-hub.

Installiamo figaro



Verifica sempre le ultime versioni su [rubygems](https://rubygems.org) - sito ufficiale³

Verifica sempre gli ultimi aggiornamenti sul readme.md di Figaro sito ufficiale⁴

figaro 1.1.1

Simple, Heroku-friendly Rails app configuration using ENV and a single YAML file

VERSIONS:

- 1.1.1 - April 30, 2015 (18.5 KB)
- 1.1.0 - January 27, 2015 (18.5 KB)
- 1.0.0 - September 17, 2014 (18 KB)
- 1.0.0.rc1 - April 17, 2014 (31.5 KB)
- 0.7.0 - June 27, 2013 (12 KB)

Show all versions (21 total) →

RUNTIME DEPENDENCIES:

thor ~> 0.14

DEVELOPMENT DEPENDENCIES:

bundler ~> 1.7

rake ~> 10.4

AUTHORS:

Steve Richert

OWNERS:

TOTAL DOWNLOADS
1,683,033

FOR THIS VERSION
477,407

Show all versions (21 total) →

GEMFILE:

gem 'figaro', '~> 1.1.1'

INSTALL:

gem install figaro

LICENSE:

MIT

REQUIRED RUBY VERSION:

~> 2.0

la gemma figaro

Aggiungiamo la gemma

codice: beginning figaro 01

³<https://rubygems.org>

⁴<https://github.com/laserlemon/figaro>

.../Gemfile

```
34 # figaro - configuration framework - imposta le Environment Variables - ENV["SECRET_PASSWORD"]
35 gem 'figaro', '~> 1.1', '>= 1.1.1'
```

Eseguiamo l'installazione della gemma con bundle ed il refresh dell'ambiente rbnb (opzionale)

terminal

```
$ bundle install
```

```
flaviobordonidev:~/workspace (master) $ bundle install
Fetching gem metadata from https://rubygems.org/
Fetching version metadata from https://rubygems.org/
Fetching dependency metadata from https://rubygems.org/
Resolving dependencies...
Using rake 11.1.2
Using i18n 0.7.0
Using json 1.8.3
Using minitest 5.9.0
Using thread_safe 0.3.5
Using builder 3.2.2
Using erubis 2.7.0
Using mini_portile2 2.1.0
Using pkg-config 1.1.7
Using rack 1.6.4
Using mime-types-data 3.2016.0521
Using arel 6.0.3
Using debug_inspector 0.0.2
Using bundler 1.12.5
Using byebug 9.0.5
Using coffee-script-source 1.10.0
Using execjs 2.7.0
Using thor 0.19.1
Using concurrent-ruby 1.0.2
Using multi_json 1.12.1
Using pg 0.18.4
Using sass 3.4.22
Using tilt 2.0.5
Using spring 1.7.1
Using rdoc 4.2.2
Using tzinfo 1.2.2
Using nokogiri 1.6.8
Using rack-test 0.6.3
Using mime-types 3.1
Using binding_of_caller 0.7.2
Using coffee-script 2.4.1
Using uglifier 3.0.0
Installing figaro 1.1.1
Using sprockets 3.6.0
Using sdoc 0.4.1
Using activesupport 4.2.5
Using loofah 2.0.3
Using rail 2.6.4
```

figaro bundle part1

```

installing figaro 1.1.1
Using sprockets 3.6.0
Using sdoc 0.4.1
Using activesupport 4.2.5
Using loofah 2.0.3
Using mail 2.6.4
Using rails-deprecated_sanitizer 1.0.3
Using globalid 0.3.6
Using activemodel 4.2.5
Using jbuilder 2.5.0
Using rails-html-sanitizer 1.0.3
Using rails-dom-testing 1.0.7
Using activejob 4.2.5
Using activerecord 4.2.5
Using actionview 4.2.5
Using actionpack 4.2.5
Using actionmailer 4.2.5
Using railties 4.2.5
Using sprockets-rails 3.0.4
Using coffee-rails 4.1.1
Using jquery-rails 4.1.1
Using rails 4.2.5
Using sass-rails 5.0.4
Using web-console 2.3.0
Using turbolinks 2.5.3
Bundle complete! 13 Gemfile dependencies, 57 gems now installed.
Use 'bundle show [gemname]' to see where a bundled gem is installed.
flaviobordonidev:~/workspace (master) $ bundle exec figaro install
      create  config/application.yml
      append  .gitignore
flaviobordonidev:~/workspace (master) $

```

figaro bundle part2

terminal

```

$ bundle exec figaro install
  create  config/application.yml
  append  .gitignore

```

L'installazione di figaro si è già preoccupata di escludere il file di configurazione da git.

[codice: beginning figaro 02](#)

.../.gitignore

```

23 # Ignore application configuration
24 /config/application.yml

```

terminal

```

$ git add -A
$ git commit -m "install figaro"

```

Cominciamo a mettere al sicuro le password

Avendo installato figaro spostiamo tutte le password sul file `/config/application.yml` Iniziamo con il `secret_key_base` che è usato per verificare l'integrità dei signed cookies e che si trova da rails 4.1 sul file `config/secrets.yml`. (Nelle precedenti versioni di rails era in `config/initializers/secret_token.rb`)

Questo file ha una "convenzione" con heroku per quanto riguarda la produzione. Infatti heroku crea in automatico la ENV["SECRET_KEY_BASE"] come puoi verificare da terminale.

terminal

```
$ heroku config
```

Quindi questa possiamo lasciarla così com'è oppure possiamo prendere quella di heroku ed archivarla sul file di figaro config/application.yml Comunque per le altre due Spostiamo i valori.

Apriamo il file ../config/secrets e copiamoci i valori dei secret_key_base

[codice: beginning figaro 03](#)

../config/secrets.yml

```
13 development:
14   secret_key_base: 2d64e44d814e3fdf518fb7f830e0f656bfcd015c86dfc4c5686d2b671a6e05aeaf67e780beaaa82b5f3be2c58bd28d616ffef2\
15 845233ab5dba9fade5067e0c06
16
17 test:
18   secret_key_base: 2de1ca50ad0877af447c1f414419e3eab1e0d09f612a6d24cd846bfcea38e3750d4965323aa35d39f945bb18b5f1592c1c590f\
19 fee3dbec39973ca299bbacb1ca
```

Passiamoli sul file di Figaro

../config/application.yml

```
1 development:
2   SECRET_KEY_BASE: 2d64e44d814e3fdf518fb7f830e0f656bfcd015c86dfc4c5686d2b671a6e05aeaf67e780beaaa82b5f3be2c58bd28d616ffef2\
3 845233ab5dba9fade5067e0c06
4
5 test:
6   SECRET_KEY_BASE: 2de1ca50ad0877af447c1f414419e3eab1e0d09f612a6d24cd846bfcea38e3750d4965323aa35d39f945bb18b5f1592c1c590f\
7 fee3dbec39973ca299bbacb1ca
8
9 production:
10  #SECRET_KEY_BASE: already present on heroku. See $ heroku config
```

**ATTENZIONE!**

le variabili d'ambiente sono case-sensitive quindi ENV["secret_key_base"] è DIVERSO da ENV["SECRET_KEY_BASE"]

Per convenzione le variabili d'ambiente si scrivono tutte maiuscole.

**DOPPIA ATTENZIONE!**

dopo le modifiche al file config/application.yml può essere necessario chiudere TUTTO IL TERMINIALE e non solo uscire dalla rails console e rientrare.

quindi il file secret.yml risulta

[codice: beginning figaro 04](#)

.../config/secrets.yml

```
13 development:
14   secret_key_base: ENV["SECRET_KEY_BASE"]
15
16 test:
17   secret_key_base: ENV["SECRET_KEY_BASE"]
18
19 # Do not keep production secrets in the repository,
20 # instead read values from the environment.
21 production:
22   secret_key_base: <%= ENV["SECRET_KEY_BASE"] %>
```

terminal

```
$ git add -A
$ git commit -m "secure passwords"
```

Giochiamo con le variabili d'ambiente (ENV)

Verifichiamo usando la rails console (<https://pragmaticstudio.com/blog/2014/3/11/console-shortcuts-tips-tricks>)

terminal

```
$ rails console
> puts ENV.keys # find out what ENV vars are set
=> (returns a long list of var names)
> puts ENV['SECRET_KEY_BASE']
=> "067d793e8781fa02aebd36e239c7878bdc1403d6bcb7c380beac53189ff6366be"
```

Riproviamolo nell'ambiente di test

terminal

```
$ rails console test
> puts ENV.keys
```

Riproviamolo nell'ambiente di produzione (con sandbox per rollback delle modifiche)

terminal

```
$ rails console production --sandbox
> puts ENV.keys
```



ATTENZIONE! l'ambiente di produzione coì chiamato fa riferimento ad un pubblicazione in locale e NON su Heroku.

Per heroku credo si debba usare un'altro comando. Ad esempio " heroku run bash "

Chiudiamo il branch

se abbiamo finito le modifiche e va tutto bene:

terminale

```
$ git checkout master  
$ git merge figaro  
$ git branch -d figaro
```

Heroku

Andiamo subito in produzione così risolviamo di volta in volta gli eventuali problemi che si presentano senza essere costretti a fare un troubleshooting su tutto l'applicativo.

Apriamo il branch

Creiamo il Branch per la pubblicazione in produzione

terminale

```
$ git checkout -b pubprod
```

Heroku piattaforma cloud PaaS

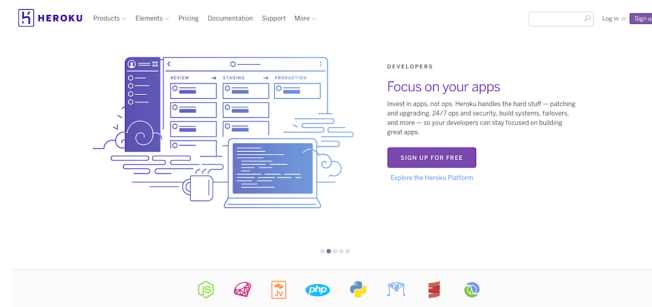
Heroku è una piattaforma cloud di programmazione progettata per aiutare a realizzare e distribuire applicazioni online. Nata nel 2007, è oggi una delle più grandi piattaforme PaaS esistenti, con centinaia di migliaia di clienti sparsi in tutto il mondo. Supporta sei linguaggi di programmazione (grazie ai quali sviluppare le app da distribuire sul web) ed è basata sul sistema operativo Debian o, più recentemente, su Ubuntu (due distribuzioni Linux).

Che cosa è una piattaforma cloud PaaS

Anche se la gran parte degli internauti crede che il cloud si risolva esclusivamente nel cloud storage, la realtà è ben altra. Il cloud computing si compone di tante branche (e il cloud storage è una di questa), raggruppate in tre macro-gruppi: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) e Software as a Service (SaaS).

Che cosa è Heroku

Heroku è stata una delle prime piattaforme a comparire sul web a permettere ai propri utenti di sviluppare, distribuire e gestire app direttamente online. Nonostante inizialmente fornisse supporto solo per progetti compatibili con l'interfaccia di programmazione web Rack, con il tempo Heroku ha ampliato la propria offerta aggiungendo centinaia di servizi e il supporto per sei linguaggi di programmazione (Java, Ruby, Node.js, Scala, Clojure, Python e PHP).



heroku home page

Grazie all'integrazione con Git, ad esempio, gli utenti potranno sviluppare la loro app su una piattaforma esterna per poi importare il progetto all'interno del proprio account Heroku, lasciando che la piattaforma PaaS faccia il resto del "lavoro sporco" per renderla operativa e funzionante.

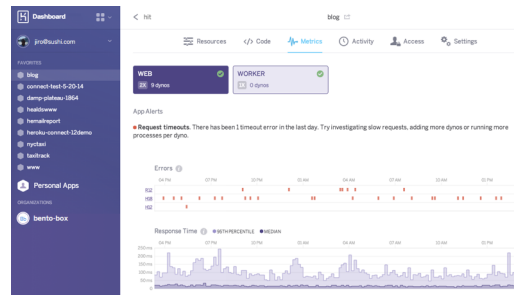
Nel 2011 si unisce al team Yukihiro Matsumoto, chief designer di Ruby; in quello stesso anno la piattaforma si arricchirà del supporto ai linguaggi Node.js e Clojure. Nel settembre 2011 nasce, dalla collaborazione tra Heroku e Facebook, Heroku for Facebook, piattaforma di servizi ottimizzata per ospitare app sviluppate per il social network di Mark Zuckerberg.

La storia di Heroku

Nata da un'idea di James Lindenbaum, Adam Wiggins e Orion Henry, Heroku fa la sua comparsa sul web nella seconda parte del 2007, mettendo a disposizione degli utenti una piattaforma sulla quale far girare applicazioni compatibili con Rack. La crescita della piattaforma è molto veloce e porta con sé molte novità: nell'ottobre 2009 al gruppo dei fondatori si aggiunge Byron Sebastian, che va a ricoprire il ruolo di CEO; l'anno successivo la piattaforma è acquistata da Salesforce.com e implementata come sussidiaria dell'azienda.

A cosa serve e come funziona Heroku

Lo scopo della piattaforma è quello di fornire agli utenti le risorse informatiche (sia hardware sia software) necessarie a distribuire e far "girare" app web-based su svariate piattaforme (ad esempio la già citata Facebook). In questo modo gli sviluppatori non dovranno preoccuparsi di avere un'adeguata infrastruttura informatica a supporto della loro creatura, ma potranno affittarla da Heroku. Non solo: la piattaforma ideata da James Lindenbaum, Adam Wiggins e Orion Henry si occupa anche del "lavoro sporco", compilando ed eseguendo automaticamente l'app non appena il codice sorgente sarà caricato nell'ambiente operativo. I programmatori potranno così concentrarsi sullo sviluppo di un codice il più possibile "pulito" e privo di bug, demandando alla piattaforma PaaS il resto dei compiti.



heroku dashboard

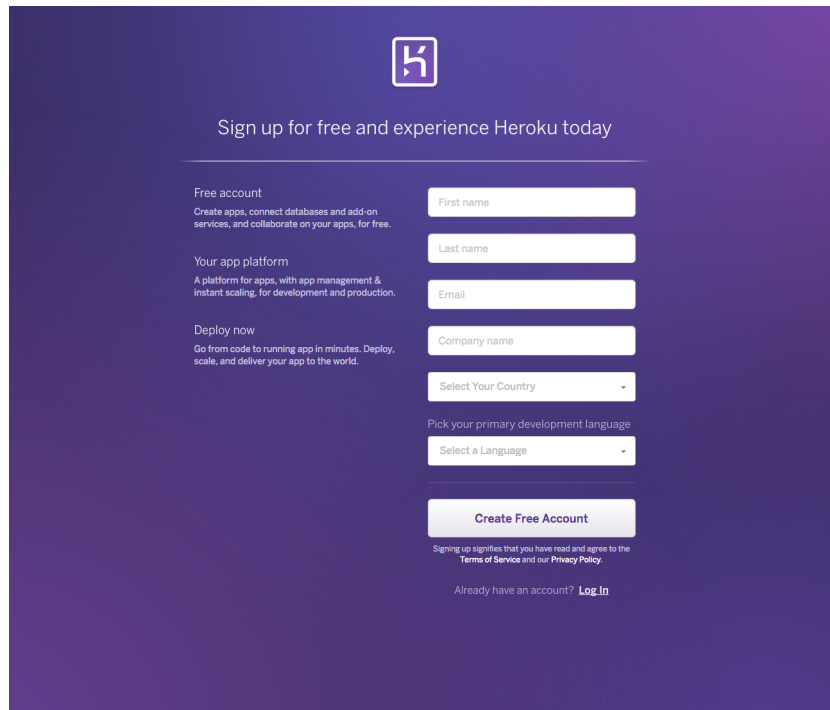
Alla base dei servizi di Heroku troviamo i dyno, container di funzioni capace di eseguire un solo comando che può essere un processo web, un processo worker (batch) o un altro tipo di processo la cui sintassi è consentita nel file Procfile. A seconda delle esigenze e della potenza di calcolo richiesta, lo sviluppatore può decidere di affittare uno o più dyno, così da assicurare ai propri utenti la miglior esperienza possibile. Affinché ciò sia possibile, Heroku offre una piattaforma completamente scalabile e automatizzata: sarà il sistema centrale a gestire l'allocazione delle risorse per ogni dyno e per ogni servizio offerto tramite l'app, sgravando il programmatore di compiti di questo genere.

Heroku come repository remoto

Heroku è quindi l'ambiente di produzione in cui viene pubblicata la nostra applicazione rails. Per caricarsi, l'applicazione, come descritto in precedenza, sfrutta git. Heroku è infatti un repository remoto di git.

Inizializziamo Heroku

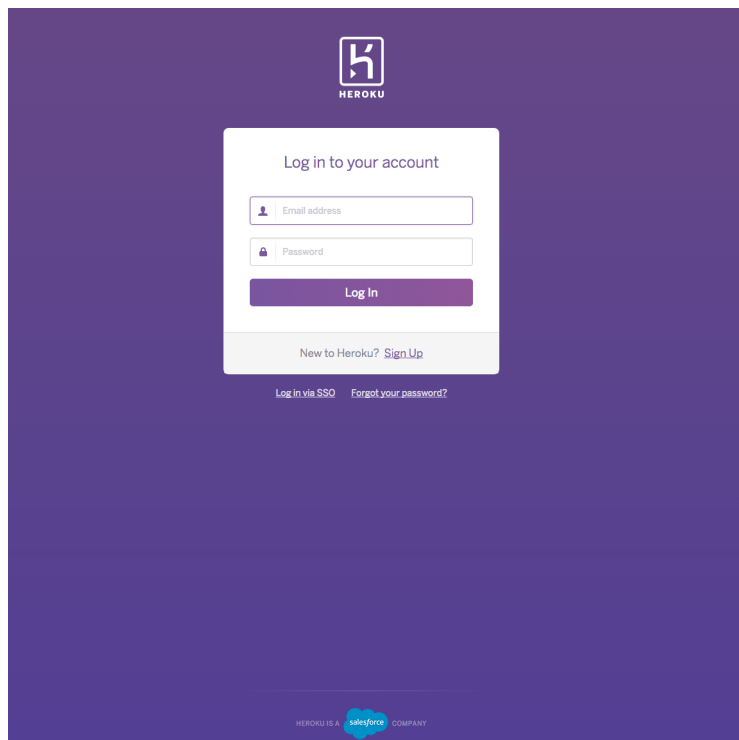
Pubblichiamo la nostra applicazione sull'ambiente di produzione Heroku. Per prima cosa creiamoci un account su www.heroku.com



The image shows the Heroku sign-up page. At the top is the Heroku logo (a stylized 'H' in a square). Below it, the text "Sign up for free and experience Heroku today" is centered. The form is divided into two columns. The left column contains three sections: "Free account" (with subtext "Create apps, connect databases and add-on services, and collaborate on your apps, for free."), "Your app platform" (with subtext "A platform for apps, with app management & instant scaling, for development and production."), and "Deploy now" (with subtext "Go from code to running app in minutes. Deploy, scale, and deliver your app to the world."). The right column contains input fields for "First name", "Last name", "Email", "Company name", "Select Your Country" (a dropdown menu), and "Pick your primary development language" (a dropdown menu labeled "Select a Language"). Below these fields is a large "Create Free Account" button. Under the button, there is a line of text: "Signing up signifies that you have read and agree to the Terms of Service and our Privacy Policy." At the bottom of the form, it says "Already have an account? [Log In](#)".

account su heroku

Logghiamoci e scarichiamo la heroku-toolbelt



The image shows the Heroku login page. At the top is the Heroku logo (a stylized 'H' in a square) with the word "HEROKU" underneath. Below the logo, the text "Log in to your account" is centered. The form is a white box with two input fields: "Email address" (with a person icon) and "Password" (with a lock icon). Below these fields is a large "Log In" button. Under the button, there is a line of text: "New to Heroku? [Sign Up](#)". At the bottom of the form, there are two links: "Log in via SSO" and "Forgot your password?". At the very bottom of the page, there is a footer that says "HEROKU IS A  COMPANY".

heroku login

Settaggio per la pubblicazione su Heroku

ATTENZIONE con RAILS 5 non c'è bisogno di questo paragrafo. C'è già la gemma puma e la gemma rails_12factor è stata integrata nel core di rails5

La gemma 'rails_12factor' è usata da Heroku per servire assets statici come le immagini e gli stylesheets. La gemma 'puma' è un web server professionale usato da Heroku. La gemma 'pg' installata nel capitolo precedente serve per gestire il database PostgreSQL.

Creiamo l'app su heroku

terminal

```
$ heroku version
$ heroku login
$ heroku keys:add
$ heroku create
```

```
flaviobordonidev:~/workspace (heroku) $ heroku create
Creating app... done, ● afternoon-lowlands-92761
https://git.heroku.com/afternoon-lowlands-92761.git
```

heroku create

Viene creato dinamicamente:

<https://enigmatic-bastion-29886.herokuapp.com/> | <https://git.heroku.com/enigmatic-bastion-29886.git>

In questo caso è stato creato “enigmatic-bastion-29886” ma può essere qualsiasi nome creato dinamicamente.

Per verificarlo si può usare il comando

terminal

```
$ heroku domains
```

o

```
$ heroku apps:info
```

```
heroku - "bobide" x Immediate x Ruby on Rails - I Ruby on Rails - I
* Starting PostgreSQL 9.3 database server
...done
bobides64:~/workspace (production) $ heroku config
== enigmatic-bastion-29886 Config Vars
DATABASE_URL: postgres://jlcrcbkybnvixu:SKT4WLErFchlHAYXWkqL1Cvrg2c2-54-235-155-172.compute-1.amazonaws.com:5432/dcsn1456mengk
LANG: en_US.UTF-8
MAX_THREADS: 1
RACK_ENV: production
RAILS_ENV: production
RAILS_SERVE_STATIC_FILES: enabled
SECRET_KEY_BASE: f6801ea895a64e841eb27c85fe8b1acd084888834ec27227e1fd6e64928f57db3c80142c6483d6821d5
ffe88abf4cc3c887cadcb4d098ec52c9190498e8cfbb
bobides64:~/workspace (production) $ heroku apps
== My Apps
enigmatic-bastion-29886

bobides64:~/workspace (production) $ heroku apps:info
== enigmatic-bastion-29886
Addons: heroku-postgresql:hobby-dev
Dynos: web: 1
Git URL: https://git.heroku.com/enigmatic-bastion-29886.git
Owner: r.desantis@integram-system.com
Region: us
Repo Size: 28 KB
Slug Size: 31 MB
Stack: cedar-14
Web URL: https://enigmatic-bastion-29886.herokuapp.com/
bobides64:~/workspace (production) $
```

apps info

Adesso imposto il MAX_THREADS seguendo il documento di heroku

terminal

```
$ heroku config:set MAX_THREADS=1
```

Adesso è tutto pronto. Posso fare il commit finale in locale e uploadare tutto in remoto.

terminal

```
$ git add -A
$ git commit -m "Add gems for production on Heroku with Puma webserver"
```

Attenzione! per pubblicare su heroku da un branch si usa un comando specifico (`git push heroku yourbranch:master`) (vedi <https://devcenter.heroku.com/articles/git>)

terminal

```
$ git push heroku pubprod:master
```

La nostra applicazione è ora in produzione su heroku. La possiamo vedere sul browser all'URL

<https://enigmatic-bastion-29886.herokuapp.com/> <https://enigmatic-bastion-29886.herokuapp.com/homepage/index>



siccome non abbiamo ancora tabelle di database è stato sufficiente fare il “git push” ma quando avremo tabelle di database dobbiamo ricordarci di eseguire il “migrate” anche su heroku.

terminal

```
$ heroku run rake db:migrate
```

Chiudiamo il branch

se abbiamo finito le modifiche e va tutto bene:

terminal

```
$ git checkout master
$ git merge pubprod
$ git branch -d pubprod
```

```
flaviobordonidev:~/workspace (master) $ heroku run rake db:migrate
Running rake db:migrate on 2 afternoons-lowlands-92761... up, run.5658
(14.8ms) CREATE TABLE "schema_migrations" ("version" character varying NOT NULL)
(5.0ms) CREATE UNIQUE INDEX "unique_schema_migrations" ON "schema_migrations" ("version")
ActiveRecord::SchemaMigration Load (1.4ms) SELECT "schema_migrations".* FROM "schema_migrations"
Migrating to DeviseCreateUsers (20160726134116)
(0.7ms) BEGIN
== 20160726134116 DeviseCreateUsers: migrating =====
-- create_table(:users)
(11.6ms) CREATE TABLE "users" ("id" serial primary key, "email" character varying DEFAULT '' NOT NULL, "encrypted_password" character varying DEFAULT '' NOT NULL, "reset_password_token" character varying, "reset_password_sent_at" timestamp, "remember_created_at" timestamp, "sign_in_count" integer DEFAULT 0 NOT NULL, "current_sign_in_at" timestamp, "last_sign_in_at" timestamp, "current_sign_in_ip" inet, "last_sign_in_ip" inet, "created_at" timestamp NOT NULL, "updated_at" timestamp NOT NULL)
-> 0.0104s
-- add_index(:users, :email, {:unique=>true})
(3.1ms) CREATE UNIQUE INDEX "index_users_on_email" ON "users" ("email")
-> 0.0069s
-- add_index(:users, :reset_password_token, {:unique=>true})
(3.4ms) CREATE UNIQUE INDEX "index_users_on_reset_password_token" ON "users" ("reset_password_token")
-> 0.0090s
== 20160726134116 DeviseCreateUsers: migrated (0.0357s) =====

SQL (2.1ms) INSERT INTO "schema_migrations" ("version") VALUES ($1) [{"version", "20160726134116"}]
(2.1ms) COMMIT
flaviobordonidev:~/workspace (master) $
```

heroku run rake db:migrate

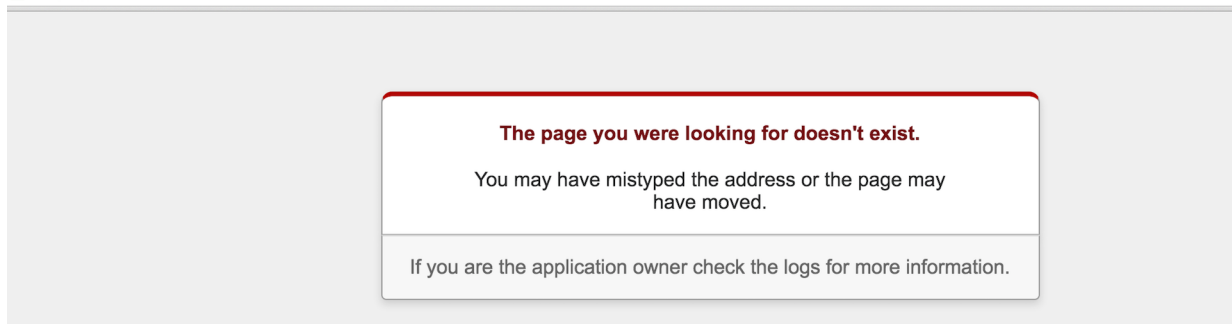
← → ↻ <https://afternoon-lowlands-92761.herokuapp.com/homepage/index>

Questa è l'homepage

il testo verrà preso dal database ma alcuni messaggi sono passati dall'applicazione ed è quindi bene che vengano tradotti per essere pronti a supportare più lingue.

heroku homepage

<https://afternoon-lowlands-92761.herokuapp.com>



heroku url root

```
➤ heroku url:root --app 92761
URL
heroku:  https://afternoon-lowlands-92761.herokuapp.com/
root:  https://afternoon-lowlands-92761.herokuapp.com/

➤ heroku url:root --app 92761 --json
{"url": "https://afternoon-lowlands-92761.herokuapp.com/"}

➤ heroku url:root --app 92761 --json --pretty
{"url": "https://afternoon-lowlands-92761.herokuapp.com/"}
```

heroku db

Adesso passiamo le variabili di 'secrets' creato in figaro direttamente in produzione su Heroku.

terminal

```
$ figaro heroku:set -e production
```

Internazionalizzazione della pagina di test.

Attiviamo l'internazionalizzazione statica

Invece di scrivere i contenuti statici (quelli non presi dal database) già tradotti, è più elegante mettere dei “segnaposto” che saranno poi usati dai vari files di traduzione nelle varie lingue. Questo permette di avere un'applicazione già pronta per essere tradotta in più lingue.

Il nome dei “segnaposto” li mettiamo in inglese per dare un'impronta world-wide al nostro applicativo che ci permetterà in futuro di coinvolgere sviluppatori da tutto il mondo.

Apriamo il branch

terminal

```
$ git checkout -b in18
```

Internazionalizzazione (i18n)

Per internazionalizzazione si intende la traduzione dell'applicazione nelle varie lingue.

L'internazionalizzazione si divide in due parti. statica = traduzione delle stringhe usate nell'applicazione. Non traduce i dati del database. dinamica = traduzione dei dati del database.

Al momento noi ci occupiamo solo di quella statica.

I18n statico con YAML

Per tradurre in varie lingue il contenuto statico della nostra applicazione (quello che non è contenuto nel database) utilizziamo il file yaml che è disponibile di default su Rails. Non c'è necessità di installare una nuova gemma. Rimane comunque la possibilità di cambiare successivamente solo il backend ed usarne uno differente invece dei files yaml. Tutto il resto dell'internazionalizzazione resta invariato.

Usiamo l'helper “t” per tutte le stringhe che dobbiamo internazionalizzare. La stringa che viene passata all'helper “t” è un segnaposto che si usa nel file yaml associandogli la stringa corretta nella lingua scelta.

.../app/views/testpages/page_a.html.erb

```
1 <h1> <%= t ".first_header" %> </h1>
2 <p> <%= t ".first_paragraph" %> </p>
3 <br>
4 <p> <%= link_to t(".link_goto_page_B"), testpages_page_b_path %> </p>
```

Se carichiamo la pagina sul browser vediamo che si visualizzano i segnaposto.

Per far apparire le descrizioni invece dei segnaposti implementiamo il backend Yaml. Iniziamo con il file en.yml perché l'inglese (en) è la lingua che viene selezionata di default. Avendo usato il “.” davanti al nome del segnaposto, per convenzione Rails cerca il segnaposto catalogato nella rispettiva view. In questo caso sotto en -> homepage -> index.

codice: [beginning-testpages_in18 01](#)

.../config/locales/en.yml

```
4 en:
5   homepage:
6     index:
7       first_header: "This is the homepage"
8       first_paragraph: "the text will be taken from the database, but some messages are passed by the application and is \
9 therefore well that are translated to be ready to support more languages."
```



ATTENZIONE! i files YAML (.yml) sono sensibili all'indentatura. Per indentare usate gli “spazi” e non i “tabs”.



Attenzione! C'è un bug e sui partials non mi funziona il “.” come previsto quando uso i partials. Ho quindi scelto di usare tutto il percorso:

.../app/views/testpages/page_a.html.erb

```
1 <h1> <%= t "homepage.index.first_header" %> </h1>
2 <p> <%= t "homepage.index.first_paragraph" %> </p>
3 <br>
4 <p> <%= link_to t(".link_goto_page_B"), testpages_page_b_path %> </p>
```

Adesso creiamo il file it.yml per implementare la lingua italiana (it).

codice: [beginning-testpages_in18 02](#)

```
.../config/locales/it.yml
```

```
4 it:
5   homepage:
6     index:
7       first_header: "Questa è l'homepage"
8       first_paragraph: "il testo verrà preso dal database ma alcuni messaggi sono passati dall'applicazione ed è quindi b\
9 ene che vengano tradotti per essere pronti a supportare più lingue."
```

Scelgo lingua di default

sulla configurazione dell'applicazione imposto la lingua di default cambiando il “locale” di default:

codice: [beginning-testpages_in18 03](#)

```
.../config/application.rb
```

```
21 config.i18n.default_locale = :it
```

può essere necessario riavviare il webserver per permettere a Rails di caricare il file it.yml

terminal

```
CTRL+C      (per stoppare)
$ rails s    (per ripartire)
```

verifichiamo sul browser <http://localhost:3000>

terminal

```
$ git add -A
$ git commit -m "set i18n static"
```

pubblichiamo su heroku

terminal

```
$ git push heroku in18:master
```

Chiudiamo il branch

se abbiamo finito le modifiche e va tutto bene:

terminale

```
$ git checkout master  
$ git merge in18  
$ git branch -d in18
```

Github

Github rende disponibile a livello web i progetti gestiti da git. Github è un repository remoto di git. Noi lo utilizziamo per avere un backup del nostro applicativo e per predisporre la possibilità di uno sviluppo multiutente.

Apriamo il branch

terminal

```
$ git checkout -b gh
```

L'importanza del README.md

Quando ci colleghiamo su github la pagina che ci presenta l'applicazione è quella scritta sul file README.md ed è quindi importante tenere aggiornato il file readme con quello che fa l'applicazione. Spieghiamo cos'è il nostro progetto e quali sono i benefici per chi lo userà.

codice: 01

README.md

```
1 # Donamat v0.1.0
2 ==
```

README.md

```
34 history:
35
36 * v0.1.0 01.01.17 creazione dell'applicazione su rails 5.0.0.1
```

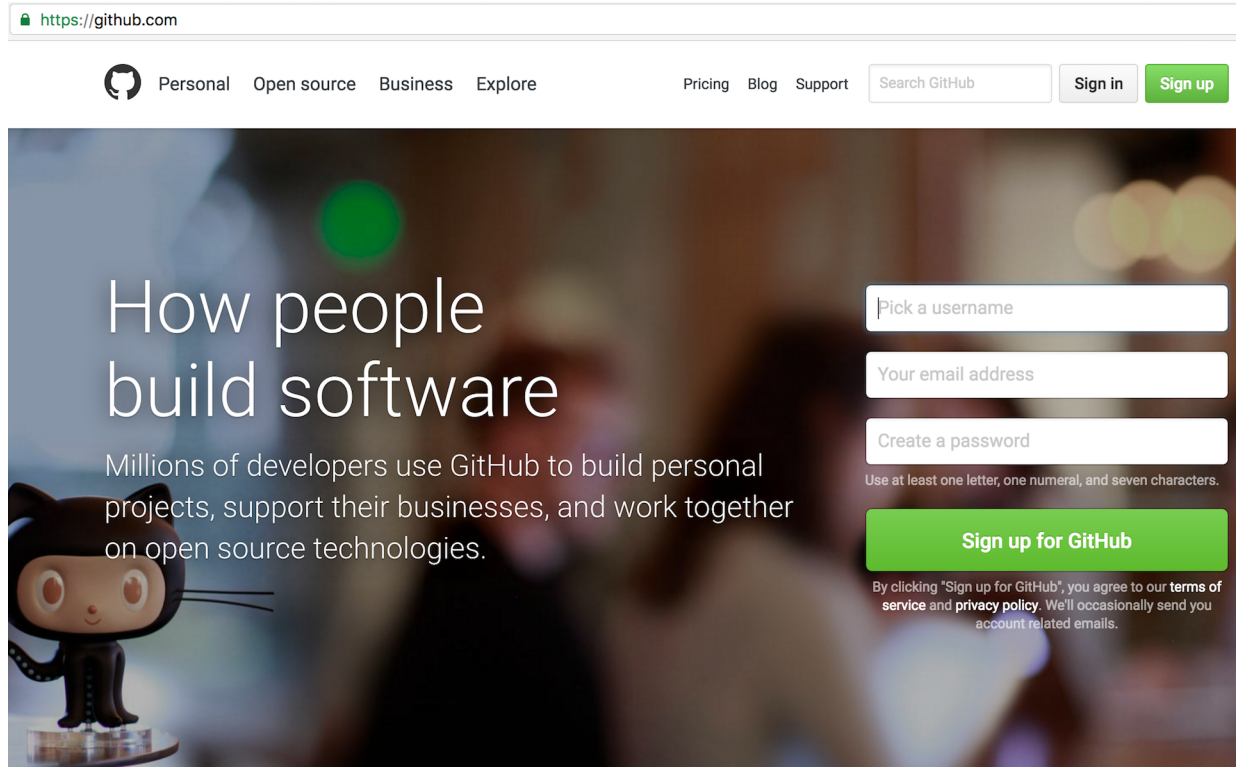
Mano a mano che sviluppiamo l'applicazione aggiorniamo anche il readme. Nello specifico la versione e l'history delle versioni.

terminal

```
$ git add -A
$ git commit -m "add readme"
```

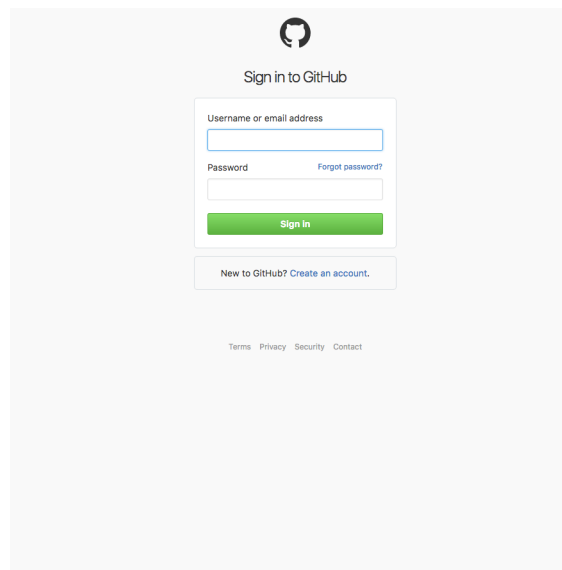
Github

Iniziamo portando la nostra applicazione su Github. Per prima cosa creiamoci un account su [www.github.com](https://github.com)



The screenshot shows the GitHub homepage with a navigation bar at the top. The main heading is "How people build software". Below it, a subheading reads: "Millions of developers use GitHub to build personal projects, support their businesses, and work together on open source technologies." On the left, there is a small illustration of the GitHub mascot, Octocat. On the right, there is a sign-up form with three input fields: "Pick a username", "Your email address", and "Create a password". Below the password field, a note says: "Use at least one letter, one numeral, and seven characters." A green "Sign up for GitHub" button is positioned below the form. At the bottom of the form, a disclaimer states: "By clicking 'Sign up for GitHub', you agree to our [terms of service](#) and [privacy policy](#). We'll occasionally send you account related emails."

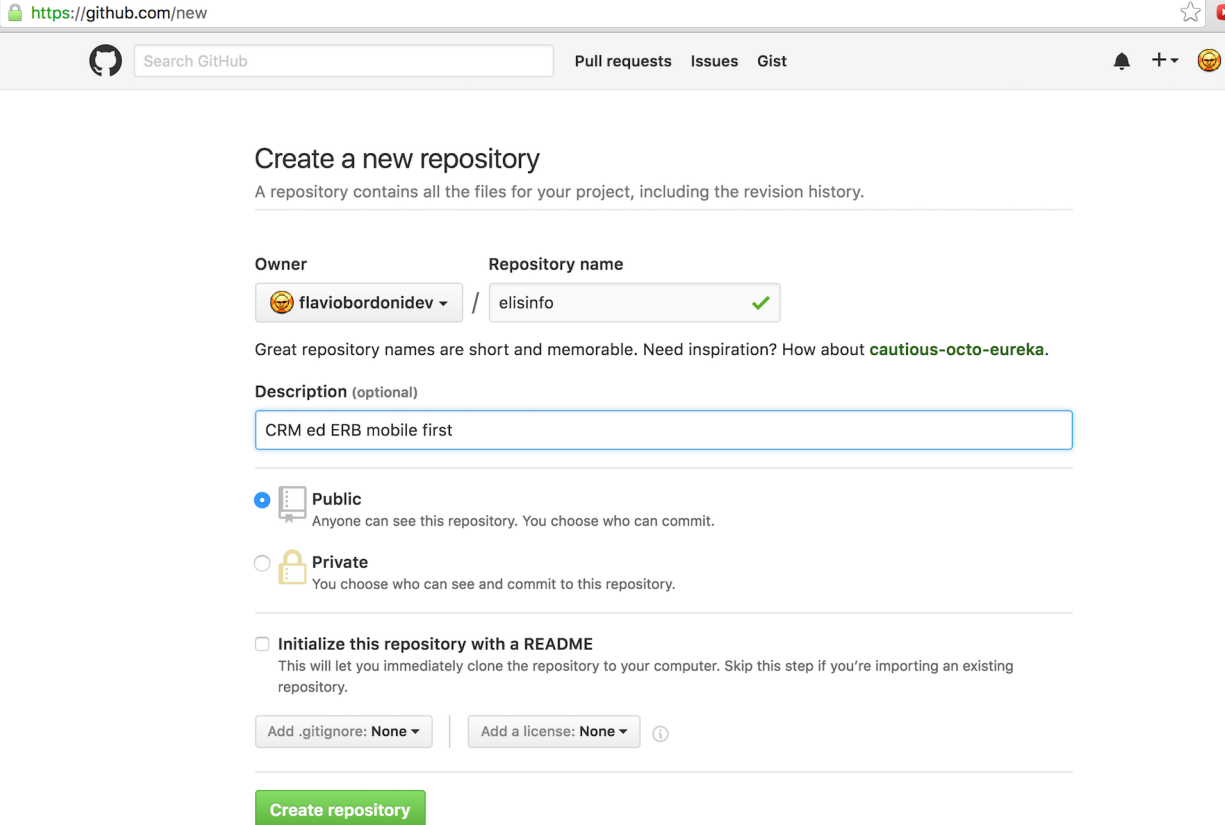
account su github



The screenshot shows the GitHub sign-in page. At the top, there is a GitHub logo and the text "Sign in to GitHub". Below this, there is a form with two input fields: "Username or email address" and "Password". A link "Forgot password?" is located next to the password field. A green "Sign In" button is positioned below the form. Below the button, there is a link: "New to GitHub? [Create an account.](#)". At the bottom of the page, there are links for "Terms", "Privacy", "Security", and "Contact".

account su github

Una volta loggati creiamo un nuovo repository e lo chiamiamo “donamat”



<https://github.com/new>

Search GitHub

Pull requests Issues Gist

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner **Repository name**

flaviobordonidev / elisinfo ✓

Great repository names are short and memorable. Need inspiration? How about **cautious-octo-eureka**.

Description (optional)

CRM ed ERB mobile first

☒ **Public**
Anyone can see this repository. You choose who can commit.

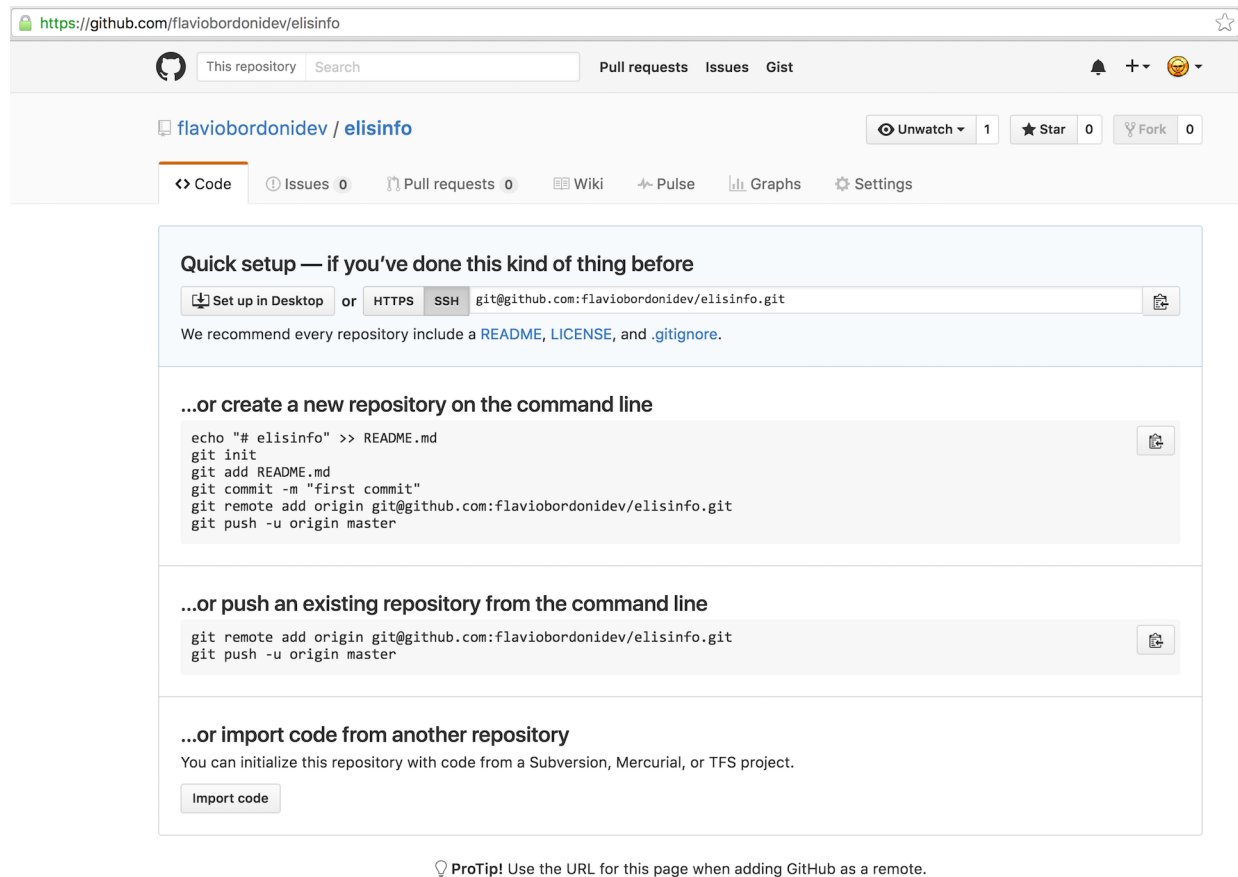
☐ **Private**
You choose who can see and commit to this repository.

☐ **Initialize this repository with a README**
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None** | Add a license: **None** ⓘ

Create repository

account su github



account su github

Aggiungiamo sul nostro git il repository remoto “donamat.git” creato sul nostro account github “Integram-System”.

terminal

```
$ git remote add origin https://github.com/Integram-System/donamat.git
```

Il comando “git remote” è per attivare il repository remoto su un server esterno (nel nostro caso github.com). con “add origin” si dichiara che il nome di riferimento del repository remoto è “origin” (potevamo chiamarlo github ma per convenzione storica la stessa Github ha scelto di chiamarlo “origin”).

Spostiamo il nostro git locale sul repository remoto Github

terminal

```
$ git push origin master
```

il comando “git push” sposta sul branch remoto “origin” il branch locale “master”. Spostiamo in remoto anche la parte dei tag in cui abbiamo messo la versione v.0.1.0

The screenshot shows the GitHub repository page for 'flaviobordonidev / elisinfo'. The repository has 3 commits, 1 branch, 0 releases, and 1 contributor. The 'Code' tab is selected, showing a list of files and their commit history. The files listed are: app, bin, config, db, lib, log, public, test, vendor/assets, .gitignore, and Gemfile. The commit history for each file is shown, with the latest commit being 'add homepage' 3 days ago.

CRM ed ERB mobile first — Edit

3 commits 1 branch 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find file Clone or download

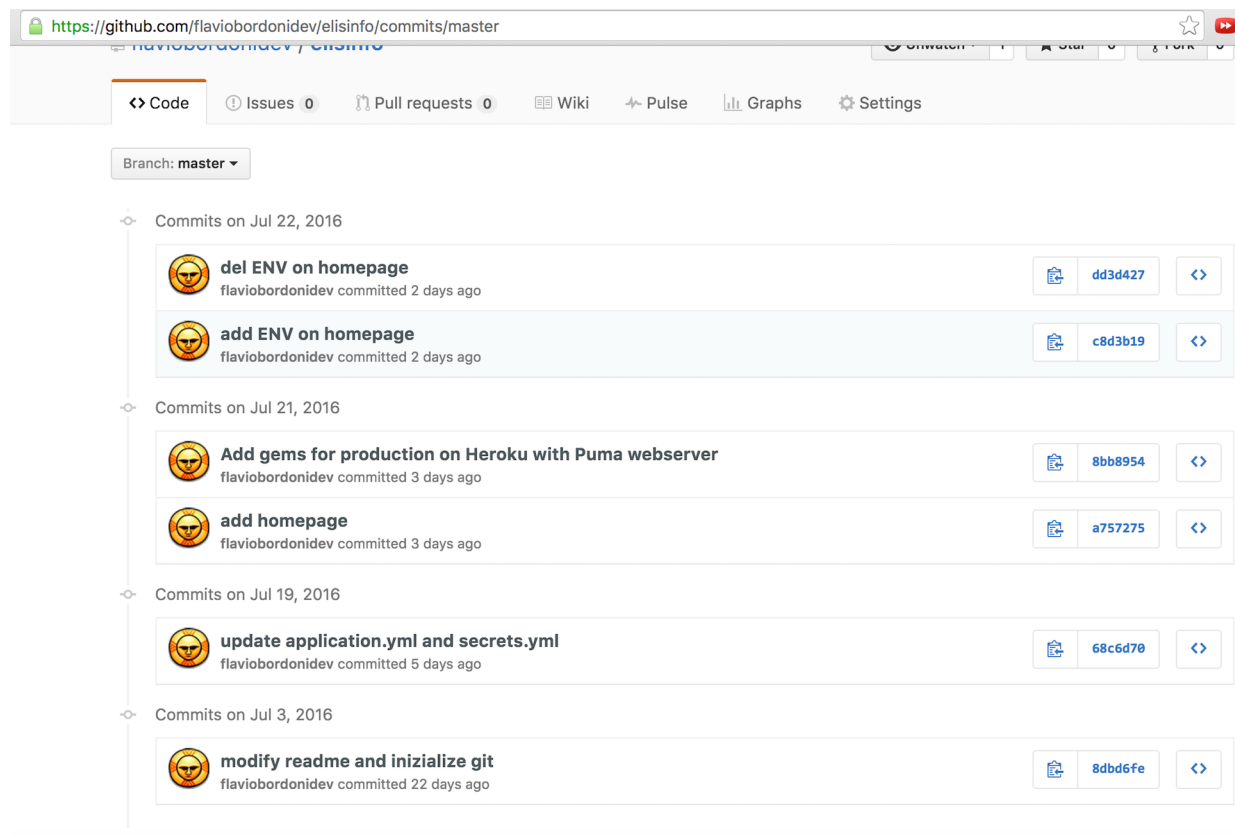
flaviobordonidev	add homepage	Latest commit a757275 3 days ago
app	add homepage	3 days ago
bin	modify readme and initialize git	22 days ago
config	add homepage	3 days ago
db	modify readme and initialize git	22 days ago
lib	modify readme and initialize git	22 days ago
log	modify readme and initialize git	22 days ago
public	modify readme and initialize git	22 days ago
test	add homepage	3 days ago
vendor/assets	modify readme and initialize git	22 days ago
.gitignore	update application.yml and secrets.yml	5 days ago
Gemfile	update application.yml and secrets.yml	5 days ago

github overview1

terminal

```
$ git push origin master --tag
```

Dopo il comando si può vedere la creazione della realease.



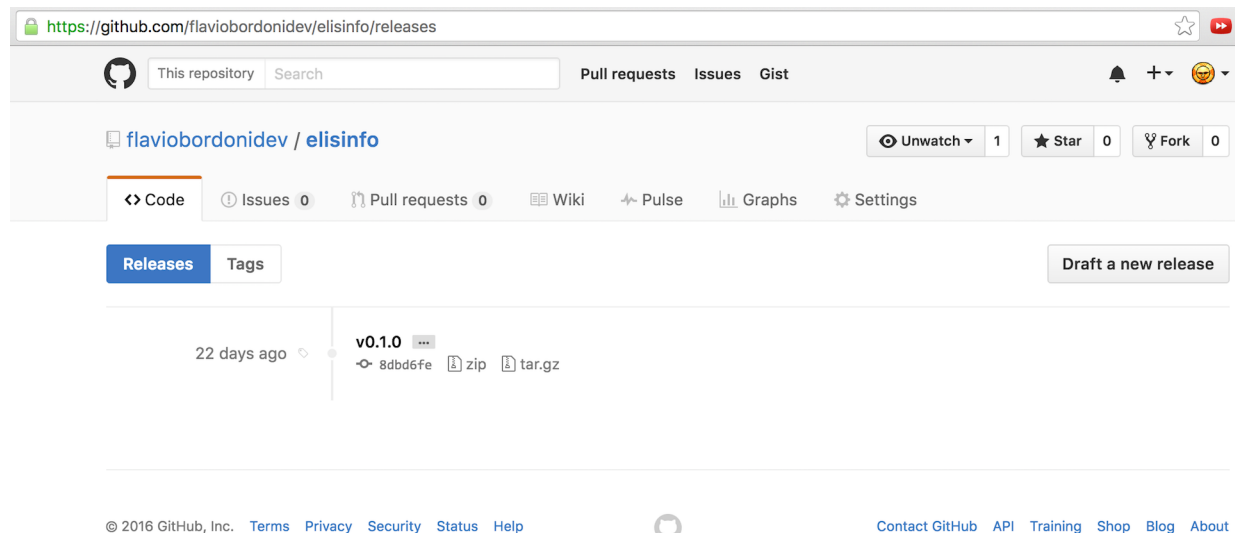
The screenshot shows the commit history for the repository `flaviobordonidev/elisinfo` on the `master` branch. The page is organized into sections by date:

- Commits on Jul 22, 2016:**
 - `del ENV on homepage` (commit `dd3d427`)
 - `add ENV on homepage` (commit `c8d3b19`)
- Commits on Jul 21, 2016:**
 - `Add gems for production on Heroku with Puma webserver` (commit `8bb8954`)
 - `add homepage` (commit `a757275`)
- Commits on Jul 19, 2016:**
 - `update application.yml and secrets.yml` (commit `68c6d70`)
- Commits on Jul 3, 2016:**
 - `modify readme and initialize git` (commit `8dbd6fe`)

Each commit entry includes the commit hash, a link to view the commit, and the author's name and commit time.

github commits

Possiamo vedere la storia dei vari commits per risalire rapidamente a tutte le modifiche storicamente effettuate.



The screenshot shows the releases page for the repository `flaviobordonidev/elisinfo`. The page displays the following information:

- Repository:** `flaviobordonidev / elisinfo`
- Actions:** Unwatch (1), Star (0), Fork (0)
- Navigation:** Code, Issues (0), Pull requests (0), Wiki, Pulse, Graphs, Settings
- Releases:** A button to "Draft a new release"
- Release List:**
 - v0.1.0** (22 days ago)
 - Commit: `8dbd6fe`
 - Assets: `zip`, `tar.gz`

The footer of the page includes the GitHub logo, copyright information (© 2016 GitHub, Inc.), and links to Terms, Privacy, Security, Status, Help, Contact GitHub, API, Training, Shop, Blog, and About.

github tags

Chiudiamo il branch

se abbiamo finito le modifiche e va tutto bene:

terminal

```
$ git checkout master  
$ git merge gh  
$ git branch -d gh
```
