Git - Pierwsze kroki

Spis treści

Instalacja

Pod tym linkiem znajdziesz instalatory **Git** dedykowane pod różne systemy operacyjne.

Windows

W przypadku gdy korzystamy w Windows, możemy również pobrać instalator znajdujący się tutaj. Możemy teraz przejść przez proces instalacji, zaznaczając same rekomendowane opcje.

Ubuntu

W przypadku korzystania z systemu Ubuntu możemy skorzystać z komendy:

```
yum install git-core
```

lub

```
apt-get install git
```

MacOS

Jeżeli natomiast jesteśmy fanami jabłuszka, możemy zainstalować **Git** wykorzystując komendę:

```
brew install git
```

Weryfikacja

Git jest narzędziem, którego można używać na dwa sposoby, z konsoli lub wykorzystując narzędzia graficzne. Wykorzystanie tego narzędzia z konsoli będzie analogiczne do tego, jak wywoływaliśmy w konsoli komendy **Maven** lub **Gradle**. Tym razem jednak komendy będą zaczynały się od słówka `git`. Jeżeli chcemy sprawdzić, czy instalacja przebiegła pomyślnie, możemy wpisać w konsoli:

```
git version
```

Jeżeli zostanie wydrukowana na ekranie informacja o wersji narzędzia - oznacza to, że instalacja przebiegła poprawnie.

Git Bash

Z gitem możesz pracować z typowej konsoli na Windows (czyli wciskasz przycisk 'Windows' i szukasz **cmd**), możesz również pracować z konsoli, która nosi nazwę **Git Bash** (u mnie jest to zlokalizowane w *C:|Program Files|Git|git-bash.exe*).

Git Bash to emulator terminala Unix, który jest częścią narzędzi Git dla systemów Windows. Git Bash umożliwia korzystanie z narzędzi Git i innych narzędzi wiersza poleceń, takich jak `grep`, `awk`, `sed`, itp. na systemie Windows w sposób zbliżony do pracy na systemach Unix.

Praca z Git z konsoli CMD na Windows jest możliwa, ale niektóre funkcje Git mogą być trudne lub niemożliwe do uzyskania z konsoli CMD. Z drugiej strony, Git Bash zapewnia pełną funkcjonalność Git i ułatwia pracę z narzędziami wiersza poleceń w sposób bardziej intuicyjny i łatwiejszy w użyciu.

Najlepiej będzie, jak popracujesz z Git z obu konsol i zobaczysz, która jest dla Ciebie wygodniejsza.

GUI

Git jest narzędziem, z którego korzystamy wpisując komendy w konsoli. Jednocześnie obsługa tego narzędzia z poziomu konsoli daje nam największe możliwości. W praktyce jednak często jest tak, że zamiast zapamiętywać wszystkie komendy, prościej jest korzystać z narzędzia graficznego. W Internecie mamy dostępnych wiele **GUI** (**G**raphical **U**ser **I**nterface) do **GIT**a.

Windows Git GUI

Na stronie możemy znaleźć poniższy cytat:

> As Windows users commonly expect graphical user interfaces, Git for Windows also provides the Git GUI, a powerful alternative to Git BASH, offering a graphical version of just about every Git command line function, as well as comprehensive visual diff tools.

Czyli korzystając z instalatora *Git for Windows*, otrzymamy również narzędzie graficzne, które pozwoli nam korzystać z **Git**a w sposób graficzny. Oczywiście nadal możemy korzystać z **Git**a w konsoli.

Sourcetree

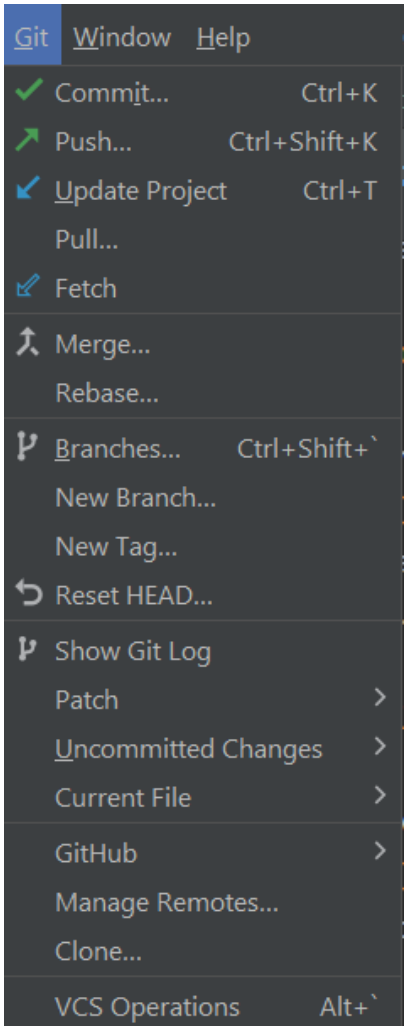
Popularnym **GUI** jest Sourcetree. Co ciekawe na stronie projektu jest napisane, że narzędzie to jest dedykowane pod Windows i Mac OS.

GitKraken

Kolejnym ciekawym **GUI** jest GitKraken. Przy tym narzędziu należy pamiętać, że jest ono płatne. Pewne funkcjonalności są udostępnione za darmo i możemy to sprawdzić tutaj.

Intellij Idea

Tak, Intellij również posiada wbudowaną integrację z **Git**em. Osobiście korzystam tylko i wyłącznie z tej integracji, nie korzystam ze wspomnianych wcześniej GUI. Intellij sam wykryje, że dany projekt wykorzystuje **Git**. Trzeba tylko najpierw nasz projekt o **Git**a oprzeć. Jak dodamy już **Git** do projektu, to pojawi nam się taka zakładka.



Obraz 1. Zakładka Git w Intellij

Trzeba tutaj zaznaczyć, że każdy programista korzysta z **Git**a jak mu się podoba. Jedni używają konsoli, inni tylko narzędzi GUI. Można też stosować hybrydę, czyli raz tak, a raz tak. Osobiście stosuję podejście hybrydowe, do prostych rzeczy korzystam z Intellij, a jak potrzebuję zrobić coś bardziej złożonego, wykorzystuję konsolę.

Konfiguracja

Udało nam się zainstalować narzędzie, ale musimy je jeszcze skonfigurować. Oczywiście możemy korzystać z ustawień domyślnych, ale zaraz zobaczysz, że jednak pewne zmiany konfiguracyjne będą Ci potrzebne. Z konfiguracją będzie związane polecenie `git config`. Jeżeli chcesz zobaczyć aktualną konfigurację narzędzia, wpisz poniższe polecenie w konsoli:

```
git config --list
```

Wyświetlą Ci się na ekranie parametry w postaci **klucz=wartość**. Klucz jest nazwą parametru, natomiast po znaczku `=` zobaczysz wartość przykładowego parametru.

Jeżeli natomiast chcesz zobaczyć jakie możliwe parametry możemy przekazać do polecenia `git config`, wystarczy, że uruchomisz poniższą komendę:

```
git config --help
```

Poziomy

W tym miejscu należy zaznaczyć, że konfiguracja **Git**a dzieli się na poziomy: **system**, **global** i **local**. Każdy z tych poziomów odnosi się do innego poziomu szczegółowości, przy czym każdy kolejny zawęża zakres widoczności tych ustawień. Oznacza to, że:

- **system** - odnosi się do najbardziej ogólnych ustawień, które są widoczne w całym systemie operacyjnym,
- **global** - odnosi się do ustawień dedykowanych dla konkretnego użytkownika systemu operacyjnego,
- **local** - ustawienia dedykowane dla pojedynczego repozytorium.

Jeżeli wrócimy teraz do komendy:

```
git config --list
```

Wyświetla ona połączone ustawienia ze wszystkich wspominanych poziomów. Oznacza to, że jeżeli na każdym poziomie pojawi się inny wpis przy tym samym kluczu, to wspomniana komenda zwróci nam duplikaty. Oznacza to, że rezultat tej komendy może zawierać zduplikowane klucze i jest to normalne zachowanie:

```
user.email=zajavka1@gmail.com
user.email=zajavka2@gmail.com
```

Natomiast z punktu widzenia ustawień konkretnego repozytorium, zostanie uwzględniona wartość na poziomie najbardziej specyficznym z dostępnych ustawień.

Poziom system

Poziom **system** odnosi się do najbardziej ogólnych ustawień, które są widoczne w całym systemie operacyjnym. Inaczej mówiąc, ustawienia te będą widoczne dla wszystkich użytkowników naszego komputera.

Ustawienia **Git**a możemy zmieniać na dwa sposoby:

- z poziomu konsoli przy wykorzystaniu komendy `git config ⋯`,
- ręcznie edytując zawartość odpowiednich plików w systemie operacyjnym.

Na każdym poziomie ogólności ustawień będziemy odnosić się do innego pliku w systemie operacyjnym. Przykładowo ustawienia na poziomie **system**, w systemie operacyjnym Windows przechowywane są w pliku:

```
C:\Program Files\Git\mingw64\etc\gitconfig
```

Zwróć uwagę, że plik ten nie ma podanego rozszerzenia, ale możemy go edytować jak plik tekstowy.

W przypadku systemu operacyjnego Ubuntu możemy poszukać analogicznego pliku w lokalizacji:

```
~etc/gitconfig
```

Możemy zatem zmienić takie ustawienia ręcznie we wspomnianych plikach. Możemy też wykorzystać komendę:

```
git config --system user.name "Roman Adamski"
git config --system user.email fakeemail@gmail.com
```

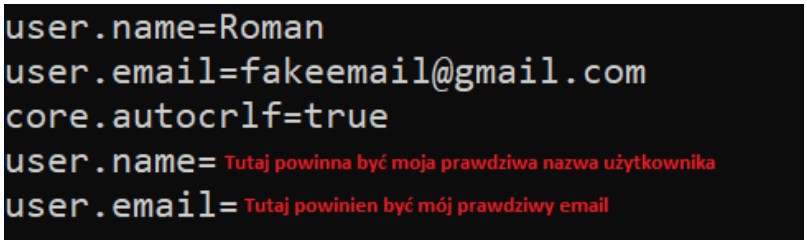
Jeżeli nie możesz uruchomić tych komend, bo dostajesz błąd: *permission denied*, musisz uruchomić to polecenie z poziomu administratora systemu operacyjnego.

Zasadniczo ten rodzaj konfiguracji jest używany dosyć sporadycznie. Wykorzystamy go wtedy, gdy będziemy chcieli ustawić to samo ustawienie dla wszystkich repozytoriów na tym samym komputerze, niezależnie od użytkownika systemu operacyjnego.

W moim przypadku, po wykonaniu powyższych komend, jeżeli uruchomię teraz polecenie:

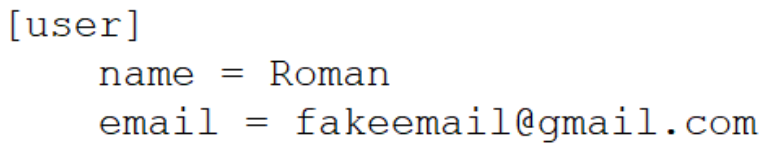
```
git config --list
```

To zostanie wydrukowane:



Obraz 2. Git config system

Adekwatny wpis pojawił się również we wspomnianym pliku konfiguracyjnym:



Obraz 3. Git config system

Mogę w tym momencie pobrać również parametry ustawień dla konkretnego klucza. W tym celu mogę wywołać komendę:

```
git config --get user.name
```

Polecenie to zwróci mi najbardziej specyficzne ustawienie dla danego klucza, jakie posiadam, czyli w moim przypadku jest to prawdziwa nazwa mojego użytkownika, która jest ustawiona na kolejnym poziomie. Jeżeli natomiast chciałbym zwrócić wszystkie możliwe wartości dla danego klucza, to wpisałbym polecenie:

```
git config --get-all user.name
```

Możemy również usunąć podane wcześniej ustawienia, w tym celu powinniśmy teraz wywołać komendy:

```
git config --system --unset user.name
git config --system --unset user.email
```

Przydatną komendą jest również poniższa, która otworzy nam bezpośrednio plik do edycji ustawień na danym poziomie ustawień. Komenda będzie bardzo przydatna, jeżeli będziemy chcieli edytować plik z ustawieniami ręcznie, ale nie będziemy mogli go znaleźć na dysku.

```
git config --system --edit
```

Poziom global

Ten poziom ustawień reprezentuje ustawienia dla aktualnego użytkownika systemu operacyjnego. Podobnie jak w poprzednim przypadku, możemy modyfikować ustawienia, wykorzystując polecenie `git config`. Zmieniamy jedynie flagę z `--system` na `--global`.

```
git config --global user.name "Roman Adamski"
git config --global user.email fakeemail@gmail.com
```

Jeżeli chcemy w tym momencie zobaczyć, jak wygląda plik z ustawieniami dla aktualnego użytkownika systemu operacyjnego, możemy wywołać poniższą komendę:

```
git config --global --edit
```

W zależności od używanego edytora tekstu można spróbować teraz przejść do lokalizacji otwartego pliku.

Poziom local

Trzeci poziom jest najbardziej specyficzny, bo jest dedykowany dla każdego repozytorium oddzielnie. Na jednym komputerze możemy mieć wiele repozytoriów lokalnych **Git** i z jednego komputera możemy się łączyć z wieloma repozytoriami zdalnymi. Aby poprawnie wywołać komendy z tym poziomem, musimy mieć stworzone repozytorium lokalne i dopiero znajdując się w nim, możemy wywołać komendy z poziomem `--local`. Jeżeli spróbuję wywołać poniższą komendę gdziekolwiek:

```
git config --local
```

To na ekranie zostanie wydrukowane:

```
fatal: --local can only be used inside a git repository
```

Żeby ten przykład miał dalej sens, sklonujmy nasze pierwsze repozytorium. Klonowanie w skrócie oznacza stworzenie kopii zdalnego repozytorium u nas na komputerze i w ten sposób otrzymamy repozytorium lokalne. Do samego klonowania jeszcze wrócimy. Wykonaj natomiast w tym momencie komendę:

```
git clone https://github.com/jhy/jsoup
```

Gratulacje! Udało Ci się w tym momencie pobrać kod źródłowy biblioteki `jsoup`. Możemy teraz w katalogu, w którym sklonowaliśmy ten projekt zobaczyć konfigurację lokalną. Przejdź zatem do utworzonego katalogu `jsoup` i wykonaj polecenie:

```
git config --local --list
```

Zobaczysz w tym momencie ustawienia z poziomu lokalnego dla tego konkretnie repozytorium. Oznacza to, że każde inne repozytorium będzie miało swoje własne ustawienia na poziomie lokalnym. Możesz tutaj również zauważyć, że jest podany adres repozytorium zdalnego, który pokrywa się z adresem, który wpisaliśmy wcześniej.

Jeżeli wykonasz w tym momencie polecenie:

```
git config --local --edit
```

Powinien otworzyć Ci się plik z ustawieniami lokalnymi dla tego repozytorium. Zwróć uwagę na lokalizację tego pliku: *.../jsoup/.git/config*. Plik ten został umieszczony w katalogu `.git`. Wspominaliśmy wcześniej o tym katalogu nazywając go **git directory**. To w tym katalogu znajduje się nasze repozytorium lokalne. Jeżeli usuniesz cały ten katalog - usuniesz całe repozytorium lokalne.

Katalog ten jest ukryty, jeżeli go nie widzisz, musisz włączyć widoczność ukrytych folderów w systemie operacyjnym.

Własne repozytorium

Lokalne repozytorium

Zostało to wspomniane wcześniej, ale napiszę to ponownie. Korzystając z **Git**a, możemy pracować ograniczając się do używania tylko repozytorium lokalnego. Nie musimy tworzyć repozytorium zdalnego. Gdybyśmy chcieli zainicjować repozytorium lokalne w projekcie, który już istnieje (czyli masz

już np. stworzony projekt w Intellij i chcesz zacząć w tym projekcie korzystać z **Git**a), wystarczy, że w katalogu z tym projektem wykonamy komendę:

```
git init
```

Na ekranie zostanie wtedy wydrukowana informacja:

```
Initialized empty Git repository in ${home_dir}/example-project/.git/
```

Gdzie zamiast `${home_dir}` zostanie wydrukowana pełna nazwa ścieżki do Twojego projektu. Jeżeli spojrzysz teraz w Intellij, to pliki na drzewie projektu (po lewej stronie), zostały zaznaczone kolorem czerwonym - wrócimy do tego później.

Przejdź teraz do katalogu z Twoim projektem i zauważ, że został tutaj utworzony katalog `.git`. Tak jak wspomniałem wcześniej, jest on ukryty. Jeżeli go nie widzisz, musisz włączyć widoczność folderów ukrytych. Cały ten **git directory** jest Twoim lokalnym repozytorium dla tego konkretnego projektu. Jeżeli w innych projektach ponownie wykonasz komendę `git init`, każdy projekt będzie miał swoje własne repozytorium lokalne, które są niezależne od siebie.

Należy zaznaczyć na tym etapie, że zostało utworzone repozytorium lokalne, ale nie ma w nim jeszcze żadnych wpisów, jest "puste". O dodanie wpisów musimy zatroszczyć się ręcznie. Jeżeli (cały czas będąc w lokalizacji Twojego projektu, czyli w katalogu *${home_dir}/example-project/*) wykonasz teraz komendę: `git log` to zobaczysz, że Twoje lokalne repozytorium nie ma dodanych jeszcze żadnych **commit**ów.

Jeżeli w tym samym projekcie masz stworzone jakieś pliki, to **Git** nie traktuje ich jak swoje, trzeba mu o tym powiedzieć. Do tego też przejdziemy.

Mając teraz stworzone takie repozytorium lokalne, moglibyśmy tworzyć w nim kolejne migawki, które dalej będziemy określali stwierdzeniem **commit**y. Kwestia jest jednak taka, że to repozytorium zostało stworzone tylko na naszym komputerze. Jeżeli pies zje nam komputer - cała historia projektu zostanie utracona.

Zdalne repozytorium

W praktyce pracuje się z repozytoriami zdalnymi. Dlatego, zanim przejdziemy do dalszej części warsztatu, chcę, żebyś przygotował/-a sobie repozytorium zdalne. W końcu materiały te mają pokazywać, jak wszystko wygląda w praktyce. 😀

Podsumowując, powiedzieliśmy sobie wcześniej o repozytorium lokalnym, czyli tym, które jest tylko na Twoim komputerze. W tym momencie zajmiemy się przygotowaniem Twojego personalnego repozytorium zdalnego, które będzie na jakimś serwerze i na którym docelowo będzie znajdowała się ta sama historia zmian, jaką będziesz wprowadzać w swoim projekcie. Gdy będziemy mieli już przygotowane repozytoria i lokalne i zdalne będziemy mogli przejść do omówienia kolejnych zagadnień.

Jak przygotujesz repozytorium zdalne i zapiszesz w nim historię zmian w projekcie, to nawet jak pies zje Twój komputer, to nadal masz replikę swojego projektu w

> repozytorium zdalnym. 😀

Trzy najpopularniejsze serwisy, które umożliwiają założenie konta i stworzenie swojego repozytorium zdalnego to GitHub, Bitbucket oraz GitLab. My w kolejnych przykładach skupimy się na używaniu **GitHub**.

GitHub

Spokojnie można powiedzieć, że **GitHub** to najpopularniejszy hosting **Git**.

> Cytując home.pl
>
> > Hosting to inaczej udostępnianie zasobów serwera, jego przestrzeni, w sieci Internet, celem umożliwienia dostępu do jego zawartości innym. Pojęciem hosting określono przestrzeń, która jest cały czas dostępna w sieci Internet, stanowi wydzieloną przestrzeń serwera, umożliwia udostępnianie online określonych zasobów, np. przestrzeni stron WWW, plików, poczty e-mail.
>
> Czyli możemy powiedzieć, że **Hosting Git** to udostępnienie zasobów serwera, żebyśmy mogli na nim stworzyć swoje repozytorium zdalne **Git**.

GitHub jest zatem taką platformą/serwerem/miejscem/przestrzenią, gdzie programiści mogą przechowywać swój kod źródłowy aplikacji. Przy wykorzystaniu **GitHub**a, programiści mogą się również dzielić kodem źródłowym swoich aplikacji z innymi. **GitHub** może być używany przez pojedynczych użytkowników, jak i przez całe zespoły.

Na moment pisania tej notatki, z **GitHub**a korzysta ponad 70 mln deweloperów i hostowanych jest ponad 200 mln repozytoriów. Należy pamiętać, że repozytoria takie zawierają projekty napisane w różnych językach programowania, więc nie dotyczy to tylko Javy. Żeby korzystać z **GitHub**a, musimy założyć tam konto. Będziemy wtedy mogli zarządzać naszymi repozytoriami z poziomu przeglądarki internetowej.

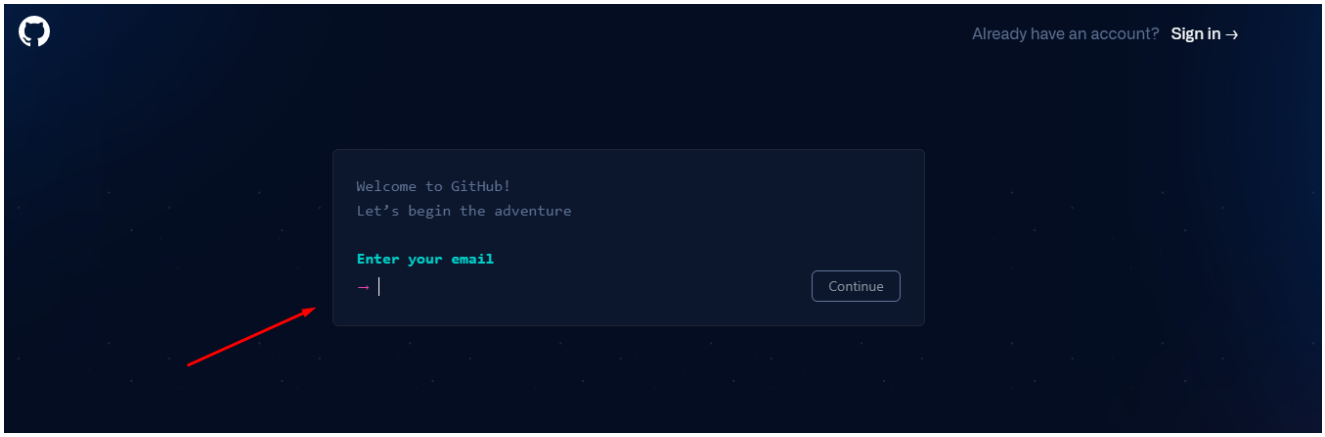
Zakładamy konto GitHub

Założenie konta jest szybkie i możemy to zrobić za darmo. Oczywiście możemy korzystać też z wersji płatnej **GitHub**, natomiast będzie to potrzebne dopiero, gdybyśmy pracowali nad większym projektem w większym zespole. Do naszych prywatnych zastosowań, wersja darmowa będzie wystarczająca. Jeżeli chcesz zapoznać się z cenami, możesz to zrobić tutaj. Mając założone takie konto, będziemy mogli dzielić się swoimi projektami z innymi programistami.

Krok1: Wejdź na stronę GitHub i kliknij **Sign up**.

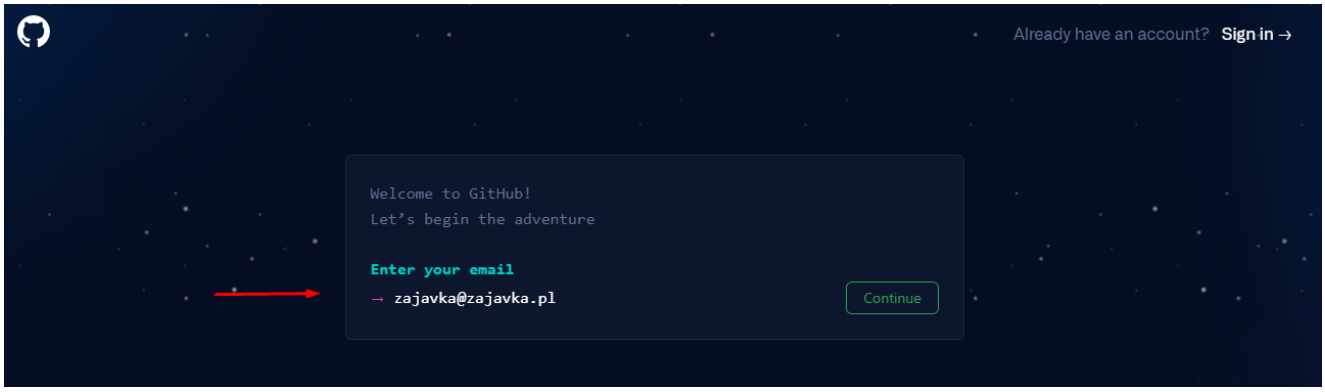
Obraz 4. GitHub krok 1

Krok2: Podaj adres email.



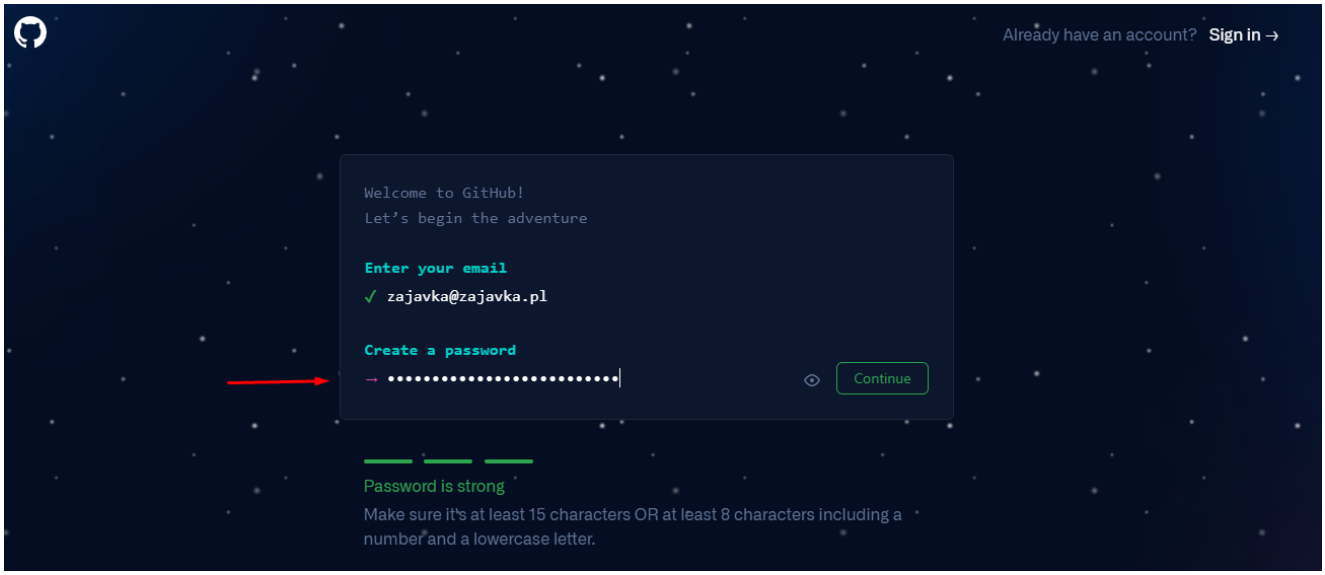
Obraz 5. GitHub krok 2

Krok3: Podaj adres email.



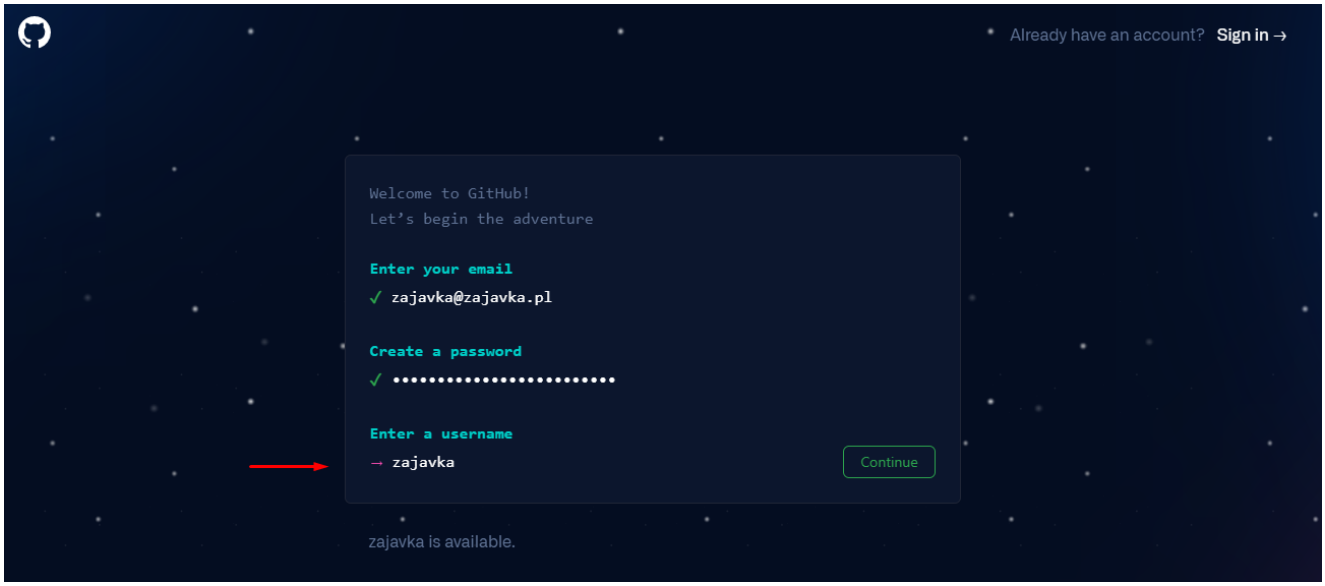
Obraz 6. GitHub krok 3

Krok4: Podaj silne hasło.



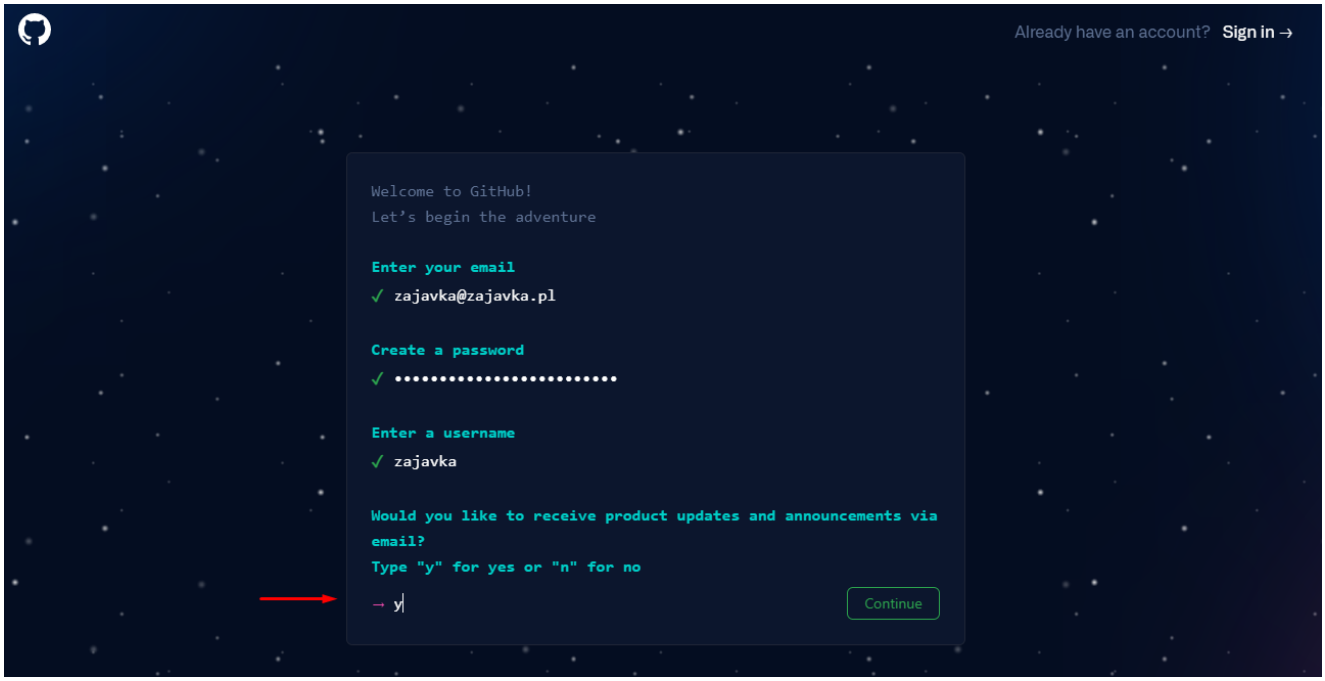
Obraz 7. GitHub krok 4

Krok5: Podaj nazwę użytkownika. Nazwa ta będzie potem używana, żeby jednoznacznie rozpoznać, że Ty to Ty.



Obraz 8. GitHub krok 5

Krok6: Wybierz **y** lub **n**. Po wybraniu tej opcji pojawi się jeszcze test czy na pewno jesteś człowiekiem.

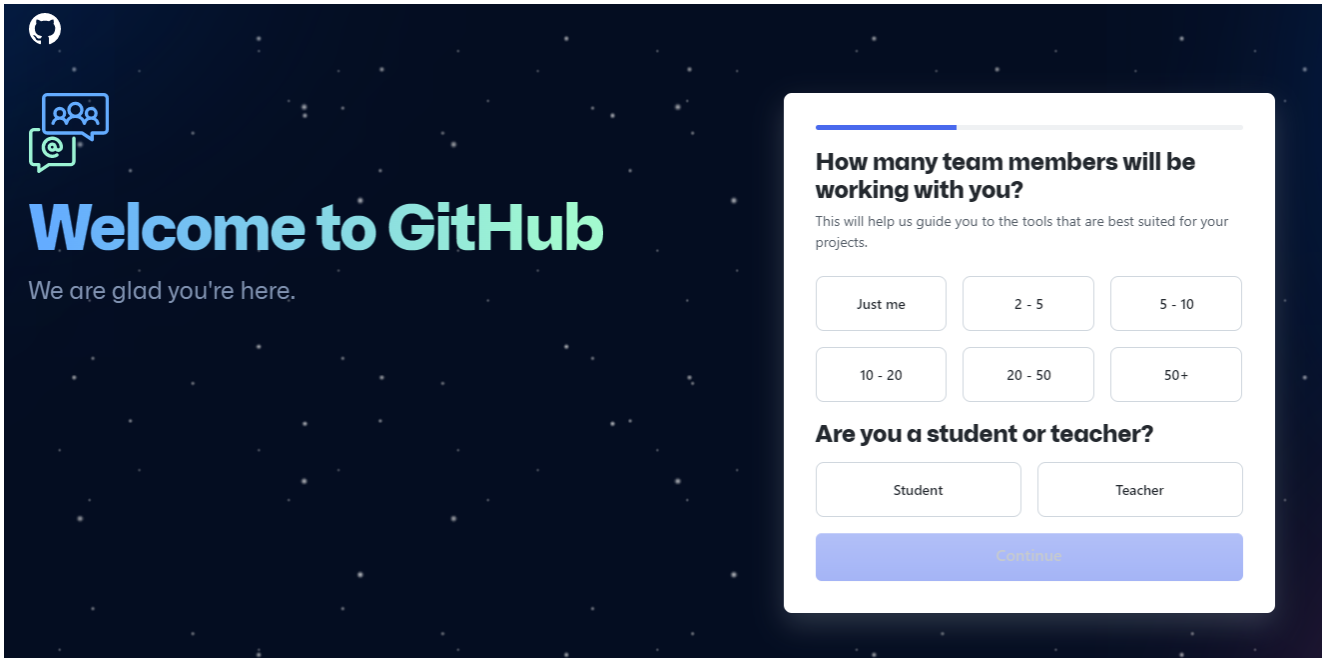


Obraz 9. GitHub krok 6

Krok7: Wpisz kod potwierdzający, który został wysłany na podany adres email.

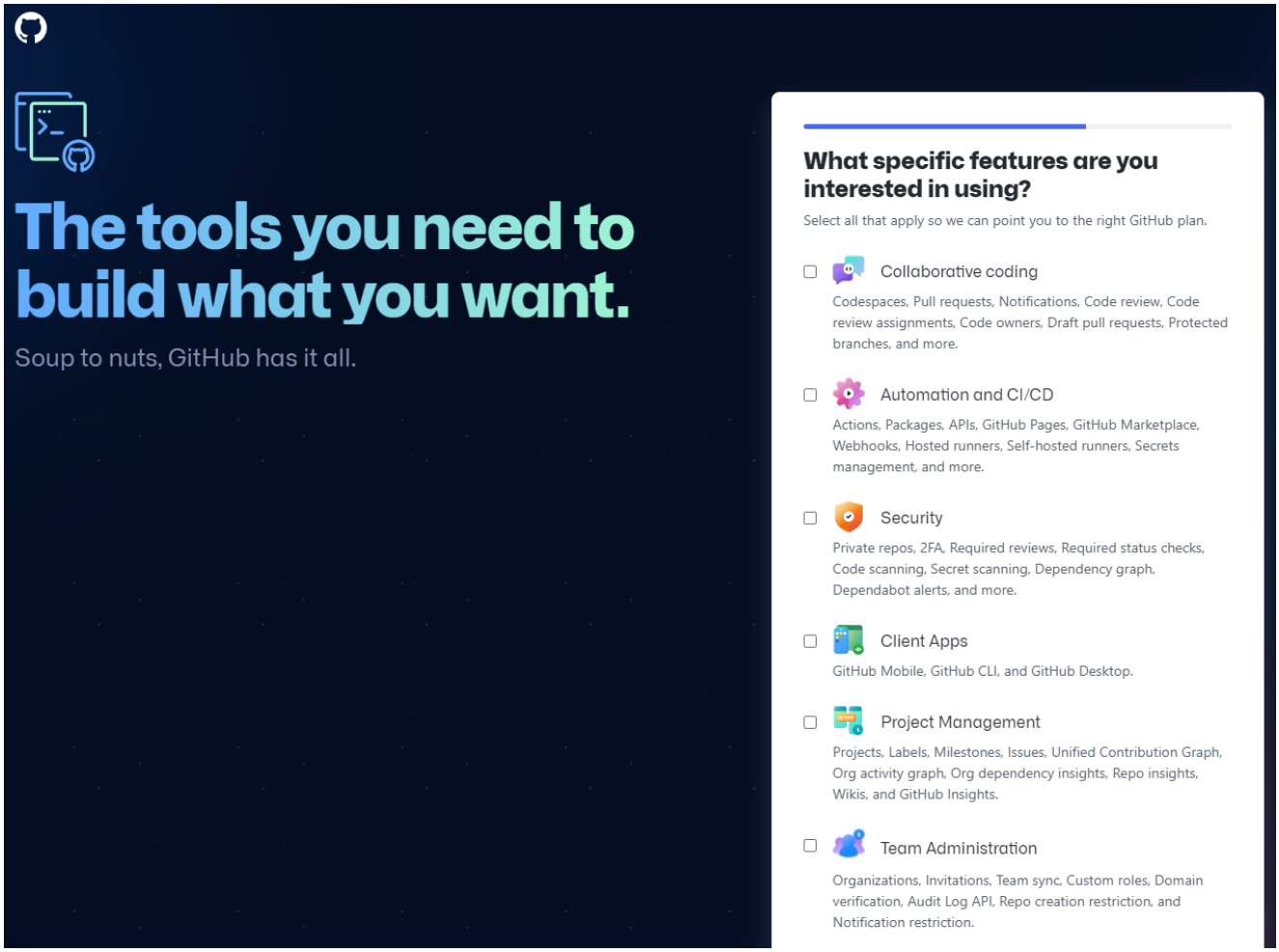
Obraz 10. GitHub krok 7

Krok8: Personalizacja. Możesz tutaj wybrać opcje zgodnie z prawdą, możesz również pominąć ten krok.



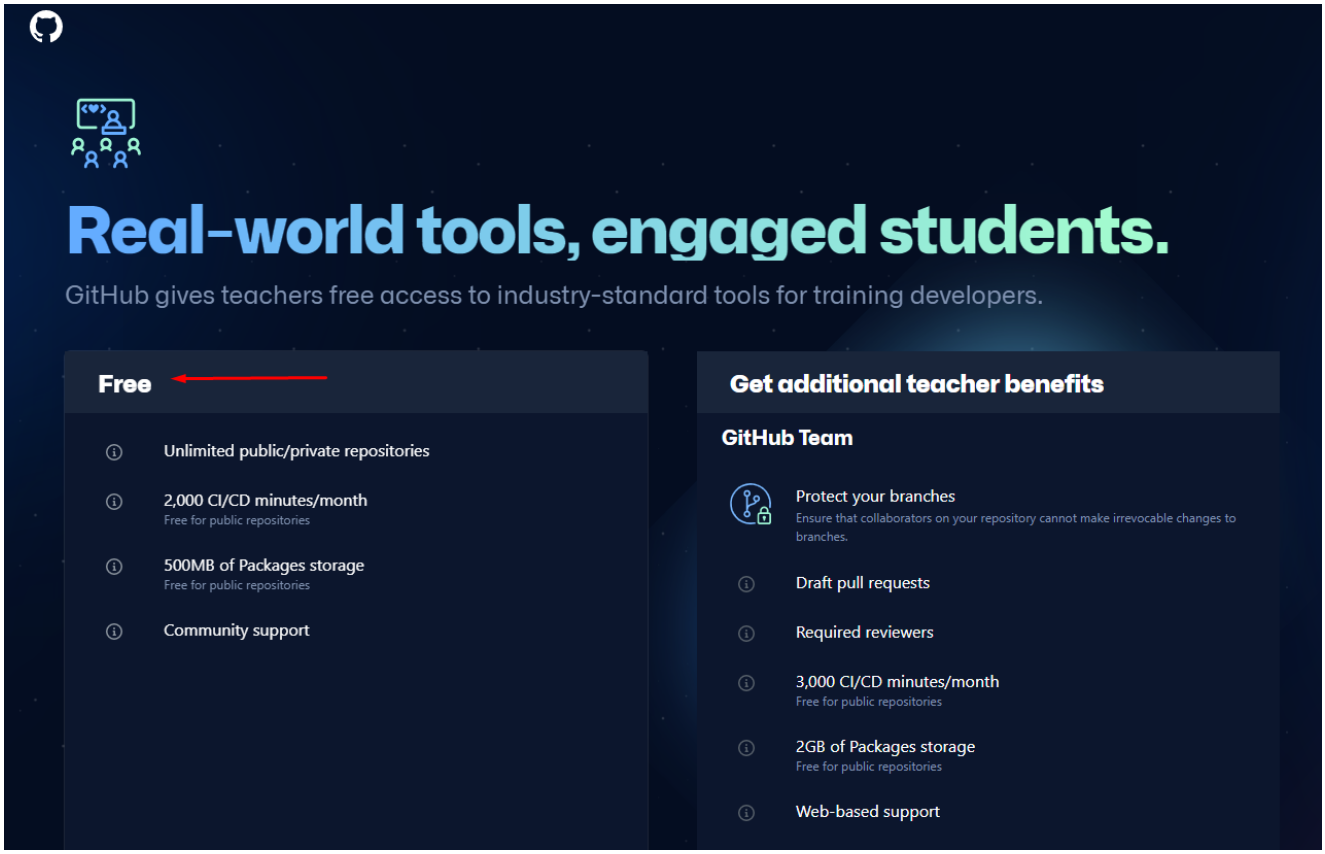
Obraz 11. GitHub krok 8

Krok9: Możesz zaznaczyć konkretne funkcjonalności, z których będziesz korzystać. Na ten moment możemy nie zaznaczać nic, możemy również pominąć ten krok.



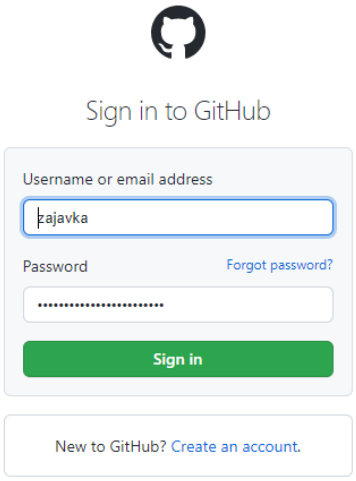
Obraz 12. GitHub krok 9

Krok10: Wybierz plan darmowy. Musisz zjechać na dół strony, tam będzie przycisk.



Obraz 13. GitHub krok 10

Krok11: Gratulacje! Od tego momentu możesz korzystać z **GitHub**. Platforma w tym momencie przekieruje Cię na stronę główną Twojego konta, którą zobaczysz do razu po zalogowaniu się. Poniższa grafika natomiast pokazuje ekran logowania.

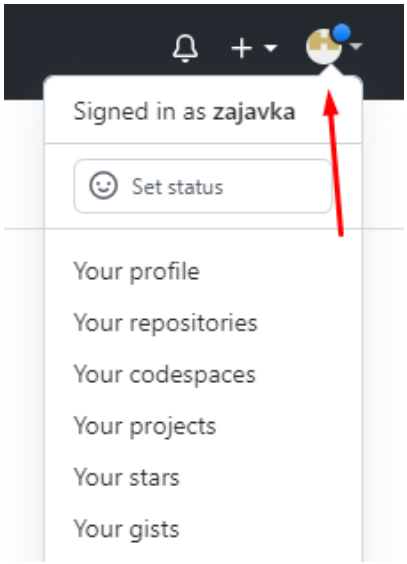


Obraz 14. GitHub krok 11

Możesz w tym momencie korzystać z desktopowej wersji **GitHub**. Możesz również pobrać aplikację GitHub Desktop, która dostępna jest na Windows.

Co po zalogowaniu?

Po zalogowaniu się na swoje konto **GitHub** możesz przejść do strony swojego profilu. W tym celu wybierz w prawym górnym rogu ikonkę z Twoim avatarem i przejdź do zakładki "Your profile".



Obraz 15. GitHub krok 12

Z tego poziomu możesz zmienić swój avatar. Na tej stronie będą też widoczne Twoje przypięte repozytoria, jeżeli sobie jakieś przypniesz. W ten sposób możesz mieć do nich szybki dostęp. Na tej samej stronie możesz zobaczyć swoją aktywność z poprzednich lat. Oczywiste jest, że jeżeli założyłeś/-aś konto przed chwilą, to będzie widoczny tylko obecny rok. ☺

Z poziomu tej platformy będziesz zarządzać Twoimi zdalnymi repozytoriami. Platforma pozwala na tworzenie nowych repozytoriów, przełączanie się między nimi oraz m.in. tworzenie nowych plików. Możliwości jest bardzo dużo i nie będę pokazywał ich wszystkich (mam nadzieję, że mi to wybaczysz). Poświęć chwilę na to, żeby poklikać po platformie, ustawieniach i powchodzić w różne zakamarki, bo dzięki temu lepiej poznasz platformę. To samo umożliwia nam aplikacja **GitHub Desktop**, ale nie będę się na niej skupiał. Jeżeli masz ochotę, przetestuj ją we własnym zakresie.

GitHub Pages

GitHub udostępnia bardzo ciekawą funkcjonalność - GitHub Pages. Idealnie nadaje się do zrobienia swojej strony internetowej, która ma służyć za Twoje programistyczne portfolio. **GitHub Pages** to serwis hostingowy, który pozwala nam udostępnić swoją stronę internetową w internecie, na podstawie statycznych plików HTML, CSS i JavaScript. Wystarczy umieścić te pliki w swoim repozytorium. Na wspomnianej wcześniej stronie GitHub Pages znajdziesz instrukcję tworzenia takiej strony internetowej. Taka strona internetowa będzie wtedy dostępna pod adresem:

```
https://<twoja_nazwa_uzytkownika>.github.io
```

Co ciekawe, przykładowo Netflix ma swoją stronę na **GitHub Pages** ⇒ https://netflix.github.io/.

GitHub Actions

Kolejną ciekawą funkcjonalnością udostępnianą przez **GitHub** jest GitHub Actions. Jak wejdziesz na podaną stronę internetową, przeczytasz wiele terminów i skrótów, których na tym etapie jeszcze prawdopodobnie nie rozumiesz. W skrócie możemy powiedzieć, że jest to platforma, która zapewnia nam zautomatyzowanie procesu budowania aplikacji (już wiemy co to, po warsztacie o Maven i Gradle),

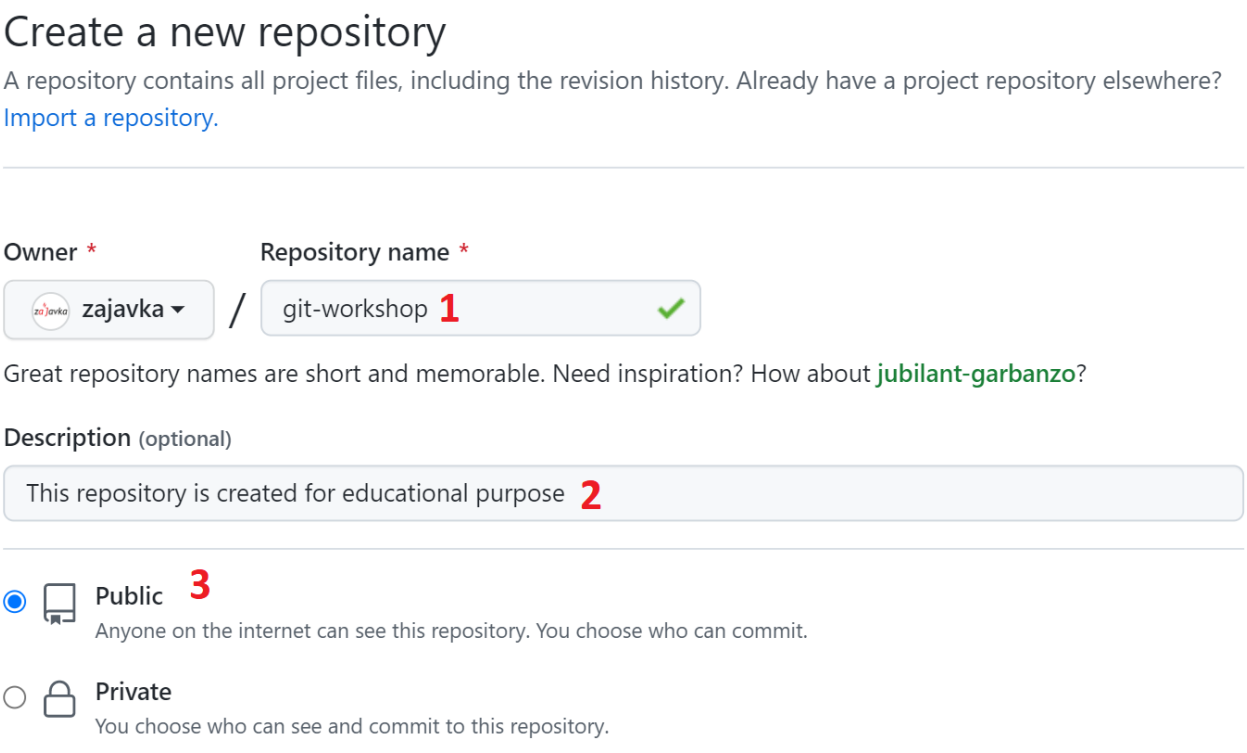
testowania aplikacji i wdrażania. Służy to do tego, żeby napisany przez Ciebie kod, w sposób automatyczny mógł finalnie trafić na środowisko produkcyjne i żeby mogli z tego kodu korzystać klienci końcowi.

Nie będziemy o tym rozmawiać w ramach tego warsztatu, natomiast warto tutaj zaznaczyć, że z pracą z **Git**em, bardzo mocno jest związane zbudowanie procesu, który pozwoli nam w sposób automatyczny wdrażać napisany przez nas kod na środowisko produkcyjne. **GitHub Actions** jest jednym z możliwych rozwiązań, ale na razie nie schodzimy głębiej w tę tematykę.

Pierwsze zdalne repozytorium

Wspomniałem wcześniej w jaki sposób można utworzyć repozytorium lokalne. Teraz przejdziemy do utworzenia repozytorium zdalnego. Wejdź na swój profil **GitHub** (czyli wejdź na stronę https://github.com/<twoja_nazwa_profilu>) i przejdź do zakładki **Repositories**. Następnie wybierz zielony przycisk "New".

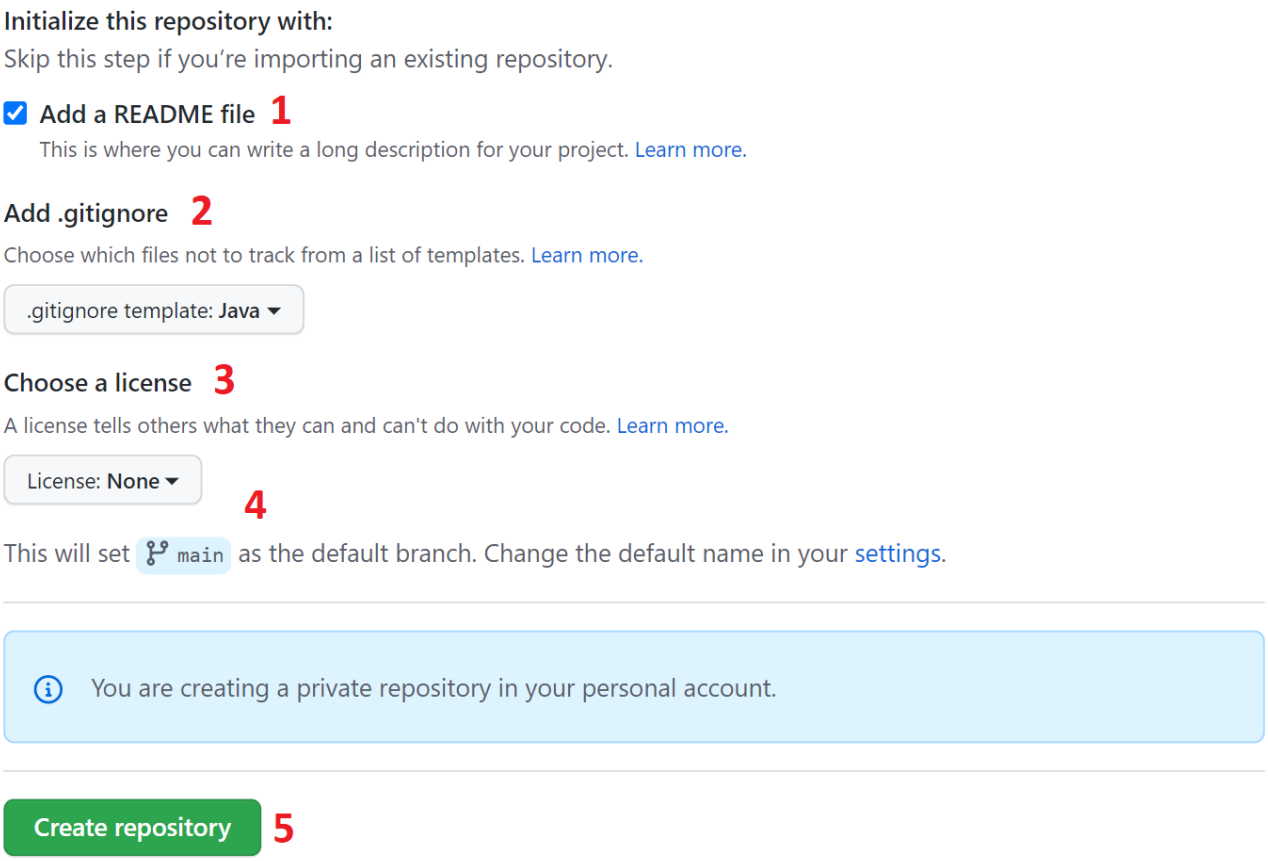
Otworzy Ci się teraz formularz pozwalający na stworzenie repozytorium.



Obraz 16. GitHub create repository part 1

1. Tutaj podajesz nazwę swojego repozytorium. Często ta nazwa jest taka sama co nazwa całego projektu.
2. Tutaj wpisujesz opcjonalny opis projektu.
3. Tutaj zaznaczasz czy repozytorium ma być prywatne, czy publiczne. Wydaje mi się, że opis pod każdą z opcji jest wystarczający. ☺

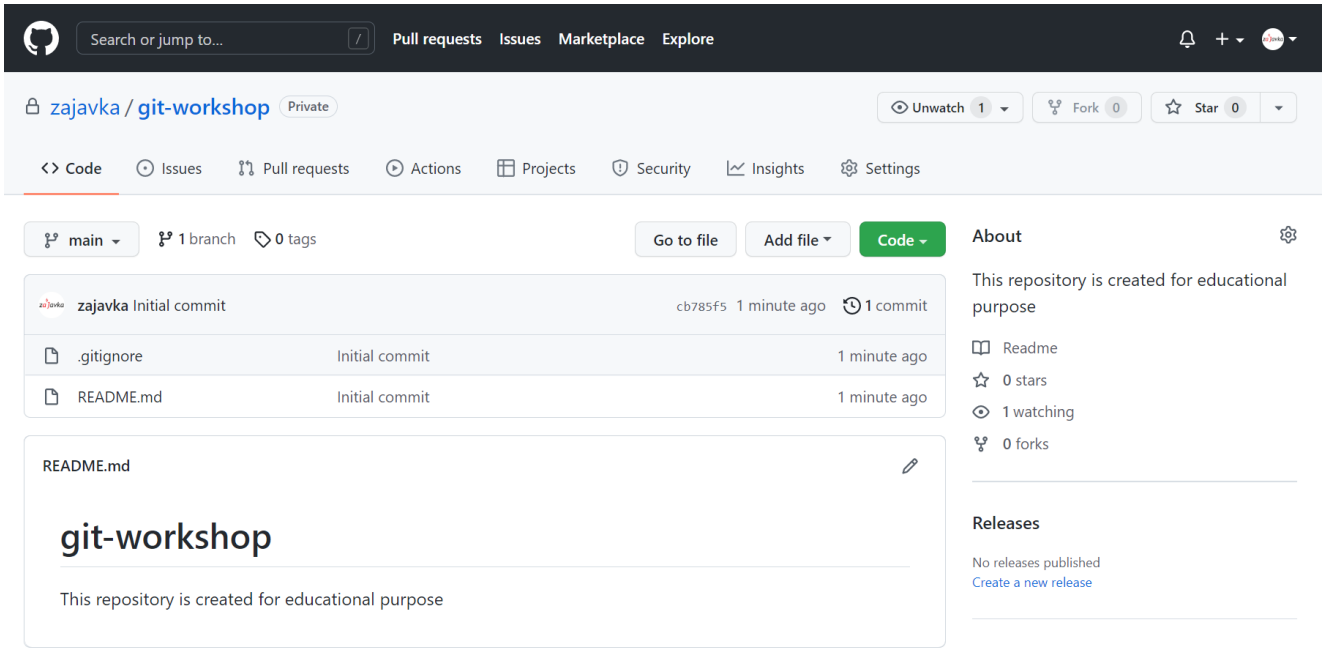
Dalsza część formularza jest opisana poniżej.



Obraz 17. GitHub create repository part 2

1. Możesz zaznaczyć tę opcję. W repozytorium zostanie wtedy automatycznie utworzony plik `README.md`, w którym możesz zamieścić opis swojego projektu. Opis ten będzie później wyświetlany na głównej stronie Twojego repozytorium w **GitHub**. Wrócimy do tego później. Jeżeli natomiast chcesz doczytać, co znaczy każda z opcji, śmiało kliknij **Learn more**.
2. Tutaj, z listy rozwijanej możesz wybrać szablon projektu Java dla pliku `.gitignore`. Wrócimy do tego pliku później. W tym momencie podpowiem, że plik ten określa jakie rodzaje plików w naszym projekcie mają być ignorowane i nieumieszczane w repozytorium.
3. Tutaj możesz wybrać licencję, na której udostępniasz swój projekt w Internecie. Wydaje mi się, że opis jest wystarczający, natomiast śmiało kliknij przycisk **Learn more**.
4. Informacja ta mówi, że stworzymy repozytorium z **branch**em domyślnym o nazwie `main`. Nic nam to nie mówi, ale o **branch**ach będziemy już niedługo rozmawiać.
5. Kliknij ten przycisk, żeby stworzyć Twoje pierwsze zdalne repozytorium.

Po utworzeniu repozytorium, na ekranie zostanie wyświetlona poniższa strona. Możemy teraz przejść do wyjaśniania kolejnych zagadnień.



Obraz 18. GitHub repository created