

# Rapport : Installation de poste

AUBIER Clément, AMARAL Paulo, BICHEL Aaron



Dans le cadre de la SAE 1.03

22/11/2022

# Table des matières

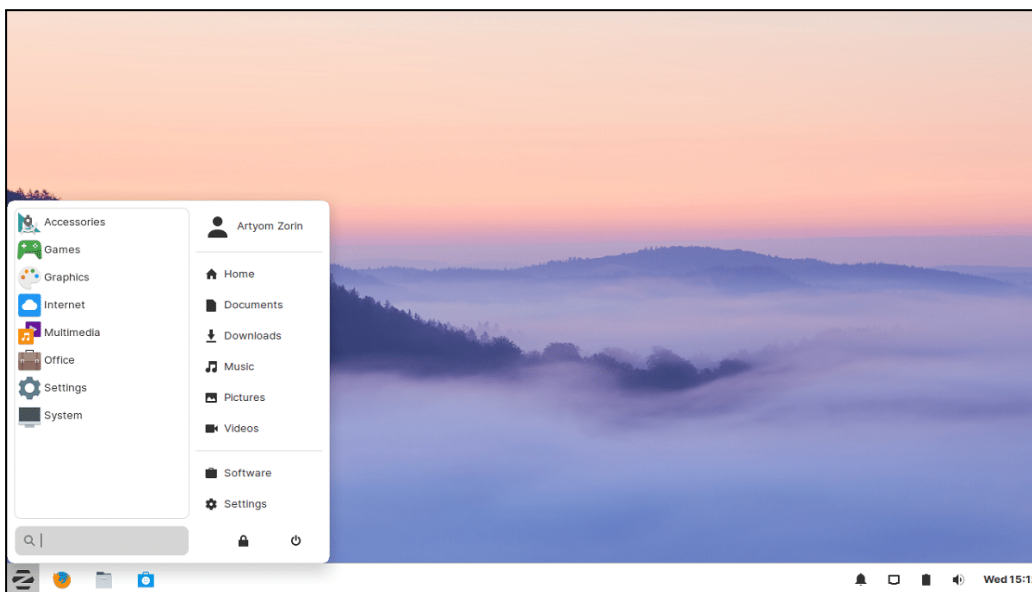
<b>Un besoin, une configuration</b>	<b>3</b>
Sélection de l'OS	3
Allocation des ressources	4
Préférences logicielles	4
<b>Organisation de la mission</b>	<b>5</b>
Architecture logicielle	5
Installation de la machine pas à pas	6
Configuration de la Machine Virtuelle	6
Installation du système d'exploitation	6
Ajout des utilisateurs	7
Installation des logiciels	8
Visual Studio Code	8
Language Go	9
Outil de gestion de versions Git	10
<b>Notice d'utilisation</b>	<b>12</b>
Comment créer un projet Go ?	12
Tester l'installation	13
Portfolio	14
<b>Conclusion du Projet</b>	<b>16</b>
Difficultés surmontées et expérience	16
Amélioration possible et limites	17

# Un besoin, une configuration

En tant qu'administrateurs système de l'entreprise BUTGAMES, il nous a été demandé de configurer les ordinateurs pour les rendre prêts à l'emploi une fois la nouvelle équipe de développeurs arrivée. Ces derniers sont chargés de programmer un jeu en Go. Ainsi nous avons préparé l'installation sur une machine virtuelle pour mieux appréhender les besoins.

## Sélection de l'OS

Tout d'abord, il a fallu choisir un système d'exploitation sur lequel les développeurs pourraient coder. Il était imposé que celui-ci devait se baser sur Linux et avoir un environnement de bureau léger. Nous avons donc choisi d'utiliser ZorinOS lite, après avoir comparé avec plusieurs distributions Linux telles que Manjaro, Linux Mint ou encore Sparky Linux. Nous avons convenu que celui-ci était le mieux équilibré en comparant les pour et les contre. Certes, cet OS n'est pas le moins lourd de ceux que nous avons comparé, mais étant basé sur xubuntu, il reste malgré tout assez léger. De plus, il utilise un gestionnaire de fenêtres basé sur XFCE, à la fois léger et assez complet pour l'utilisation qu'en fera un développeur de jeux. En complément de tout cela, cet OS est agréable à utiliser, sobre et confortable ; que l'on vienne de Windows ou d'un autre Linux, on n'est pas dépaycé.



## Allocation des ressources

Maintenant que nous avons choisi l'OS, il faut s'occuper de choisir les caractéristiques et la puissance des futures machines. Aucune demande spécifique n'a été faite de ce côté-là. Ainsi, nous avons choisi la version lite de Zorin OS car cette dernière est amplement suffisante pour l'utilisation qu'en fera un développeur. Les prérequis minimum conseillés par l'équipe Zorin OS est un processeur de seulement 1 cœur cadencé à 1 GHz, 1 Go de RAM et 10 Go d'espace disque. Cependant, nous avons décidé d'augmenter ces performances à la création de la machine virtuelle afin que cette dernière soit plus fluide. En effet, la RAM est la mémoire vive où sont stockées les informations des applications en cours d'utilisation. Nous avons donc d'abord testé avec 3Go de RAM, mais il y avait des ralentissements lors de l'utilisation d'applications. On a donc décidé d'allouer à la VM 4Go de RAM et 2 cœurs afin de pouvoir faire un minimum de multi-tâches. Nous avons ensuite créé un disque dur dynamique de 20Go pour que le développeur ait suffisamment de place pour stocker ses projets.

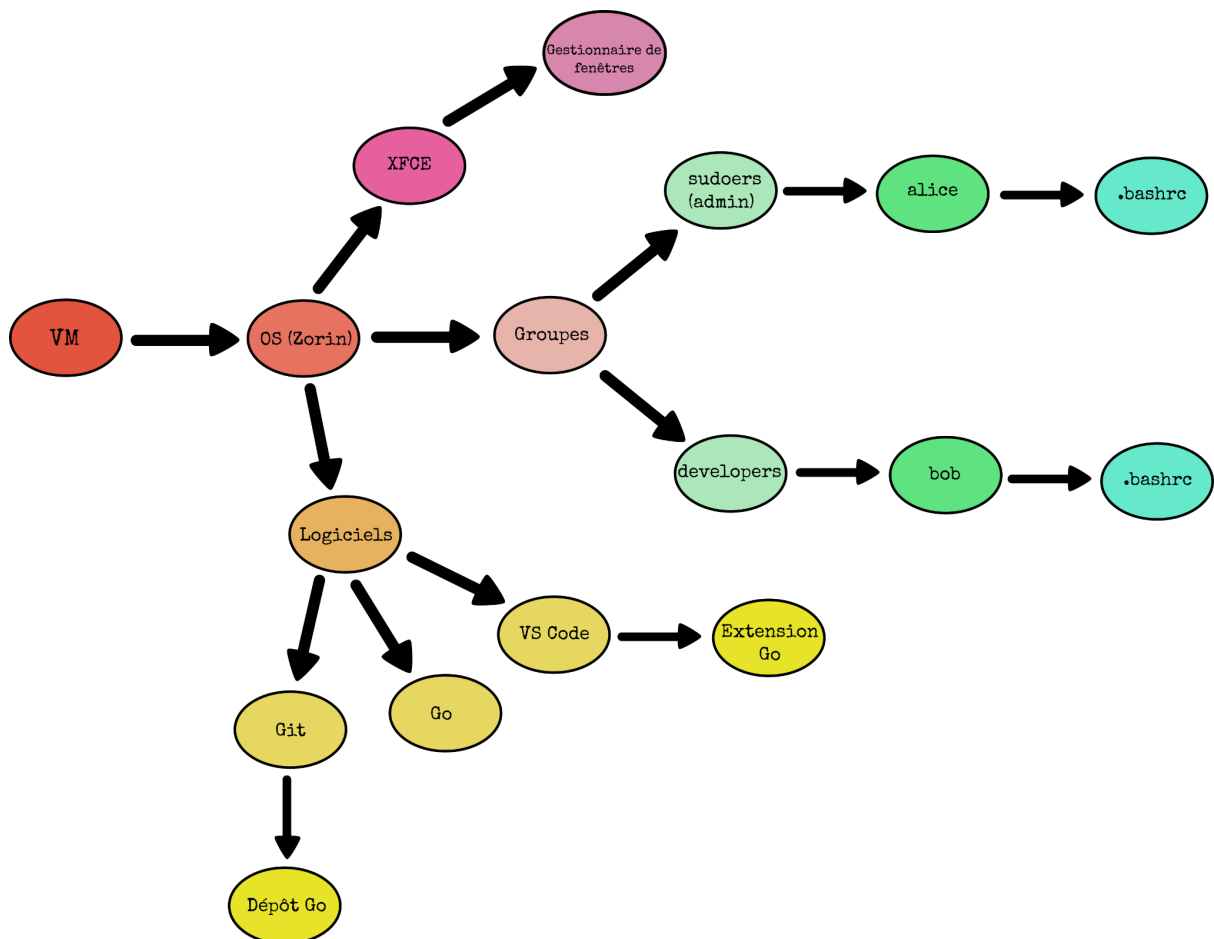
## Préférences logicielles

Concernant l'IDE installé sur la machine, nous avons respecté le cahier des charges en installant Visual Studio Code. C'est un logiciel de développement léger et complet, compatible avec le langage utilisé par les futurs développeurs de l'équipe. Au passage, nous avons également installé l'extension *Go* sur VS Code, qui améliore l'autocomplétion, facilitant ainsi le travail des développeurs. Ensuite, nous avons installé le langage Golang sur le système afin que les développeurs puissent coder en Go. De plus, il a fallu installer le logiciel de gestion de versions git afin que l'utilisateur puisse collaborer et partager facilement son travail avec les autres membres de l'équipe.

# Organisation de la mission

Maintenant que le cahier des charges est établi et que nous avons clairement en ligne de mire ce que doit comporter l'installation, nous pouvons récapituler cela en un schéma décrivant la liaison entre les différentes composantes essentielles de la configuration de la machine. Enfin, nous pourrions passer à la pratique en décrivant les différentes étapes pour arriver à paramétrer le système de cette manière.

## Architecture logicielle



# Installation de la machine pas à pas

Voici une suite d'instructions simples vous permettant de configurer vous-même la machine pour qu'elle réponde à la demande.

## Configuration de la Machine Virtuelle

Tout d'abord, commençons par télécharger l'image ISO de Zorin OS sur leur [site officiel](#). Ce dernier fait 3Go et nous permettra d'installer l'OS sur la machine virtuelle. Ensuite, il faut ouvrir VMware et cliquer sur Create a New Virtual Machine. Dans la fenêtre qui s'ouvre, cliquez sur **Sélectionner disc\_image file (iso)** et cherchez le fichier précédemment téléchargé. Si VMware ne reconnaît pas l'OS présent dans l'image de disque, il faut l'indiquer dans l'étape suivante en sélectionnant "Other Linux 5.x and later kernel 64-bit". Faites **Next** et donnez un nom à votre machine puis sélectionnez un emplacement où l'enregistrer. De plus, on alloue 4Go de RAM, 20Go de ROM et 2 coeurs à celle-ci.

## Installation du système d'exploitation

Lancez la machine virtuelle et appuyez sur "entrée" pour lancer Zorin OS. Cliquez ensuite sur **Install Zorin OS**, en n'oubliant pas de sélectionner la langue anglaise si cette dernière ne l'est pas par défaut. Dans le menu **Keyborad Layout** qui s'affiche, choisissez **French**. Après avoir cliqué sur **continue**, nous vous conseillons de laisser les options par défaut sur le menu **Updates et other software**. Dans le menu suivant **Installation Type**, choisissez **Erase disk and install Zorin OS**.

Ensuite, choisissez votre fuseau horaire et faites **continue**. Dans les champs qui apparaissent, mettez Alice dans le champ **Your name**. Choisissez un nom pour votre machine, qui vous permettra de l'identifier, notamment si vous la mettez en réseau. Choisissez **alice** comme nom d'utilisateur, qui sera utilisé dans le terminal notamment. Mettez **alice@but** comme mot de passe et continuez. Vous n'avez plus qu'à attendre que l'installation soit terminée !

Install

Who are you?

Your name: Alice ✓

Your computer's name: zorinOS ✓  
The name it uses when it talks to other computers.

Pick a username: alice ✓

Choose a password: alice@but Weak password

Confirm your password: alice@but ✓

☐ Log in automatically

☒ Require my password to log in

☐ Use Active Directory

You'll enter domain and other details in the next step.

Back Co

## Ajout des utilisateurs

Maintenant que l'OS est installé, nous pouvons nous occuper de la création des utilisateurs. Pour cela, nous devons d'abord créer un groupe développeurs, qui contiendra l'ensemble des utilisateurs du même statut. Pour cela ouvrons un terminal et tapons la commande **sudo groupadd developers**.

Ensuite, nous pouvons créer l'utilisateur bob, qui sera dans le groupe développeur, en utilisant: **sudo useradd -g developers bob -m**. L'option **-m** permet de spécifier que l'on veut également créer un répertoire personnel associé à ce profil : c'est l'espace de travail de bob. Ensuite, ajoutons-lui un mot de passe de session avec: **sudo passwd bob**, puis en saisissant le nouveau mot de passe **bob@but**.

Enfin, connectons-nous en tant que bob en saisissant **sudo su bob**, puis tapons **chsh** pour modifier le shell de connexion par défaut de bob de sh à bash. Pour ce faire, taper **/bin/bash** dans le champ de saisie qui s'affiche.

```
alice@zorinOS:~$ sudo groupadd developers
[sudo] password for alice:
alice@zorinOS:~$ sudo useradd -g developers bob -m
alice@zorinOS:~$ sudo passwd bob
New password:
Retype new password:
passwd: password updated successfully
alice@zorinOS:~$ sudo su bob
$ chsh
Password:
Changing the login shell for bob
Enter the new value, or press ENTER for the default
Login Shell [/bin/sh]: /bin/bash
$
```

## Installation des logiciels

### Visual Studio Code

Passons maintenant à l'installation des différents logiciels qui rendront la machine adaptée à l'utilisation qu'en feront les développeurs. Commençons par installer Visual Studio Code. Pour cela, rendons-nous sur le [site officiel](#) de l'application et téléchargeons le fichier d'installation **.deb**.

Tapons ensuite **sudo -s** pour se connecter en tant que root et ainsi installer le logiciel sur toutes les sessions. Placez-vous ensuite dans le dossier où vous avez téléchargé l'installateur via la commande **cd**, puis faites **sudo apt install ./nomFichier.deb**. Le fichier devrait se nommer de la sorte: **code\_1.73.1-1667967334\_amd64.deb**.

Ensuite, il faut ajouter VS Code comme éditeur par défaut. Pour cela, il faut éditer la variable d'environnement EDITOR. Rendons-nous dans le fichier **/etc/profile**, qui se charge à chaque lancement de session, et éditons-le. Faisons ainsi **nano /etc/profile** et ajoutons la ligne suivante à la fin du script: **export EDITOR=/usr/bin/code**. Faites CTRL + S puis CTRL + X pour enregistrer et quitter.

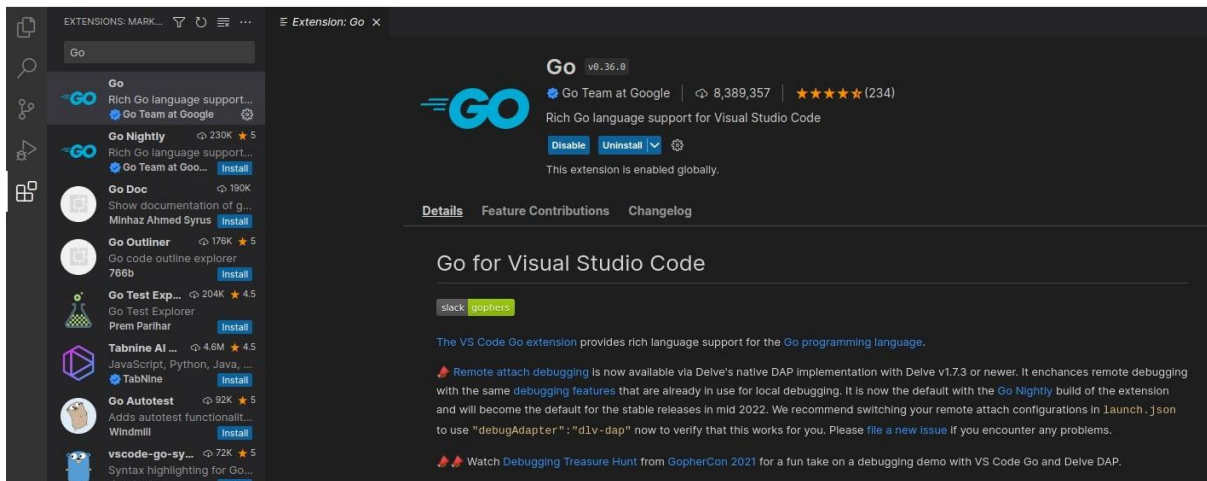


## Language Go

Nous pouvons maintenant configurer le système pour Go. Pour cela, téléchargeons le fichier d'installation **.tar.gz** sur le [site officiel](#) de Go. Faisons de nouveau la commande **sudo -s** pour nous connecter en tant que root et installer sur toutes les sessions. Plaçons-nous dans le même répertoire que le fichier téléchargé et tapons la commande qui permet de supprimer les possibles anciennes versions de Go existantes sur le système et de l'installer: **rm -rf /usr/local/go && tar -C /usr/local -xzf nomFichier.tar.gz**. Le fichier d'installation devrait se nommer de la sorte: **go1.19.3.linux-amd64.tar.gz**.

Maintenant, il faut ajouter Go à la variable d'environnement PATH afin que les binaires de l'application soient accessibles depuis n'importe où dans le terminal. Modifions encore une fois le fichier **nano /etc/profile** et ajoutons-y à la fin la ligne suivante **export PATH=\$PATH:/usr/local/go/bin**. Il faut ensuite relancer la session pour que les modifications soient effectives. Tapez ensuite **go version** dans un terminal pour voir si Go a bien été installé. Si vous souhaitez que la commande **go** ne soit accessible que depuis la session de bob, vous pouvez ajouter cette même ligne précédente, mais dans le fichier **.bashrc** de bob. Pour cela, tapez **nano /home/bob/.bashrc**, toujours en tant que root, puis écrivez-y à la fin **PATH=\$PATH:/usr/local/go/bin**.

Après s'être connecté sur la session de bob, nous pouvons installer les derniers outils de développement Go avec la commande : **go install -v golang.org/x/tools/gopls@latest**. Enfin, nous pouvons également installer l'extension Go sur VS Code, afin de rendre l'édition des projets plus agréable par les développeurs. Pour cela, lancez VS Code depuis le menu d'applications. Ensuite, cliquez sur **Extensions**, dans le volet Explorateur à gauche de l'écran. Tapez **Go** dans la barre de recherche et installez la première extension qui apparaît dans la liste, celle développée par Go Team At Google.



## Outil de gestion de versions Git

Pour terminer, il faut installer Git, afin que les développeurs puissent versionner, déposer et partager leur travail en ligne. Pour cela, faisons **sudo -s** pour installer sur toutes les sessions, puis **sudo apt install git**. Après avoir tapé **exit** pour se déconnecter de root, faire **git --version** pour s'assurer que le logiciel s'est bien installé.

Ensuite nous allons cloner le dépôt git qu'utilisera le développeur. Pour ce faire, connectons-nous en tant que bob avec **sudo su bob** et plaçons-nous ensuite dans le répertoire home de bob (on fait **cd**). Il va maintenant nous falloir générer une clé ssh pour que bob puisse s'authentifier et cloner le dépôt. Pour cela, tapons **ssh-keygen** et pressons entrée jusqu'à la fin de la procédure. Nous pouvons faire la commande: **cat /home/bob/.ssh/id\_rsa.pub** pour afficher la clé d'authentification publique de bob, qu'il pourra copier dans son compte Gitlab, dans la section Préférences > SSH Keys.

```

alice@zorinOS:~$ sudo su bob
bob@zorinOS:/home/alice$ cd
bob@zorinOS:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/bob/.ssh/id_rsa):
Created directory '/home/bob/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/bob/.ssh/id_rsa
Your public key has been saved in /home/bob/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:T+s2QXTtB+YDc7y6StYHJV0prXkNP2tkmnB8U+48cIA bob@zorinOS
The key's randomart image is:
+---[RSA 3072]-----+
|
|.o+=. |
|E*o* |
|. + @.o |
|+ .X.o |
|S+...=oB. |
|ooooo.B= |
|.o.o.*o= |
|.o o o. |
|.o . |
+---[SHA256]-----+
bob@zorinOS:~$ cat .ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQGDQTy20E00DsPidqRzjaQDhMVNyMZ8reESzRnV1VwGZzZDKZdqNrLERFSRFZE6gPJv6xUFKcX6Cm68EH/97P3BN++h08Rruj8u70TVXiWHM+8Q5LJHULDFARw
90NzskH+Lmmtvj2gJ79H+p8KpixRtiJgguxh+PVEGjTlpYeUvhWAB830LoKutWdUL9CQpgpbs4foC9MFwLrNpd1r884V6gDALh5JN+k1SU0wQm1SrJ4ARJJYXvFka4DyK2P96IIi0dmv6eeAAZrztUdPba40
W0VKSIH3iv9e6uu/0o6ISJQ+mFtIXMEHOCB++2k4aTVgS7UCpngcR1Is86sMFbPJYMcDtl6bYPSgy19ru1cAq0xfSmbAAptHsnJy/QS6EAXH0e3QPxa5UZ6W1hekBV8lcX63fPq23K0Qt9jQgaPdZ16TyrdI8M
q1oL10nzolZKhEE+06dEABdHHwsb61DIZKsCo5ZErlvPTz4XoX18rygCqgSvNIJTPESdIjnloXqE= bob@zorinOS

```

Ensuite, rendons-nous sur le [dépôt suivant](#). Cliquez sur la bouton bleu **clone**, en haut à droite, puis copiez l'URL en dessous de **Clone with SSH**. Ouvrez un terminal et faites **git clone --branch go leLien** pour cloner uniquement la branche qui concerne le développement en Go qui nous intéresse. Le lien devrait être le suivant: [git@gitlab-ce.iut.u-bordeaux.fr:rgiot/hello.sae4.but1.iut.git](https://gitlab-ce.iut.u-bordeaux.fr:rgiot/hello.sae4.but1.iut.git).

Pour que ça affiche les informations concernant le dépôt à chaque fois que bob ouvre son terminal, il faut se connecter en tant que bob avec **sudo su bob** puis modifier le fichier **nano .bashrc**. Rajouter la commande suivante à la fin du fichier: **git -C hello.sae4.but1.iut/status**. Le contenu de la commande après entre l'argument **-C** et **status** est le chemin vers le dépôt précédemment créé. Ici, ce dernier a été créé à au niveau du répertoire personnel de bob.

# Notice d'utilisation

L'installation est maintenant terminée, et le système prêt à l'emploi. Voici quelques étapes rapides afin que les développeurs puissent prendre en main facilement la configuration. Vous trouverez également dans cette section, l'expérience que nous a apporté ce projet.

## Comment créer un projet Go ?

Ci-dessous, voici les étapes permettant de créer un projet Go avec l'installation fournie.

- 1) Créez un répertoire où vous rangerez vos projets Go par la suite. Vous pouvez le faire en utilisant le gestionnaire de fichier graphique ou avec le terminal et la commande *mkdir nomFichier*.
- 2) Ouvrez VS Code depuis l'onglet des activités ou en tapant *code* dans un terminal.
- 3) Une fois VS Code ouvert, allez dans l'onglet *File*, puis *Open Folder*. Sélectionnez alors le dossier que vous venez de créer.
- 4) Cliquez sur *New Folder* dans le volet Explorateur et nommez le dossier du nom de votre nouveau projet.
- 5) Cliquez sur *New File* dans le volet Explorateur pour créer un fichier dans le dossier de votre projet et nommez-le *main.go*.
- 6) Ouvrez un terminal en faisant Terminal > New Terminal, puis saisissez *go mod init nomDossierProjet*.
- 7)
- 8) Collez le texte suivant de votre fichier *main.go*:

```
package main

import "fmt"

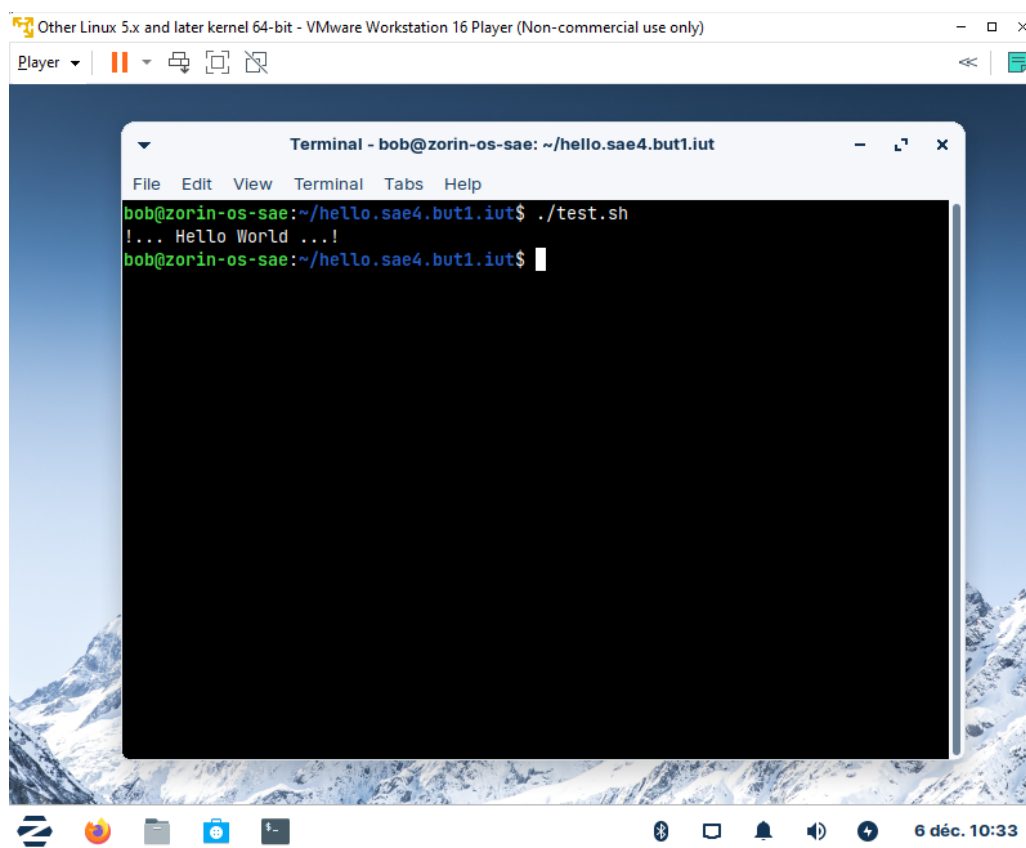
func main() {
    name:= "world"
    fmt.Println("hello", name)
}
```

- 9) Vous pouvez compiler et exécuter le projet en tapant dans la console *go run main.go*. Vous pouvez uniquement compiler votre projet en tapant *go build main.go*. Pour voir toutes les opérations faisables sur votre projet, faites *go help*.

## Tester l'installation

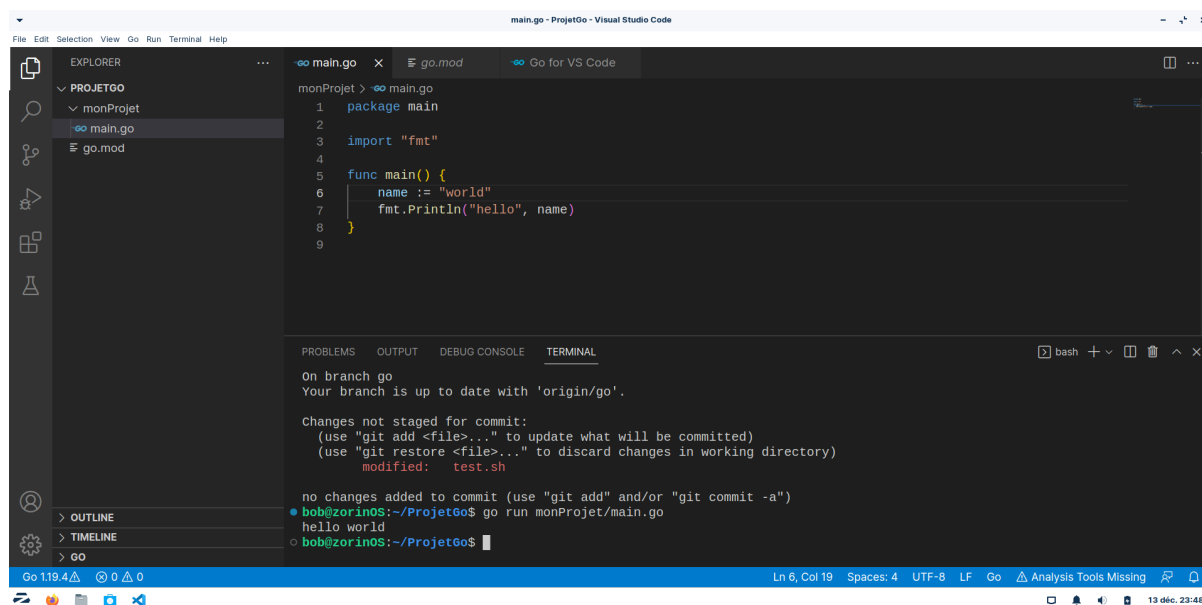
Nous pouvons également tester la configuration pour voir si cette dernière fonctionne comme prévu. Pour ce faire, on peut se connecter en tant que bob en utilisant l'interface graphique, et cloner la branche du [dépôt git suivant](https://gitlab-ce.iut.u-bordeaux.fr:rgiot/hello.sae4.but1.iut.git) via la commande *git clone -branch go git@gitlab-ce.iut.u-bordeaux.fr:rgiot/hello.sae4.but1.iut.git*. Ensuite, on peut se placer dans le dossier créé avec la commande *cd*, puis lancer le script via la commande *bash test.sh*.

Si tout se passe bien, cela devrait normalement afficher “hello world” dans la console. Sinon, il est fort probable que vous ayez oublié d'ajouter la commande “go” à variable d'environnement PATH, permettant de lancer un binaire depuis n'importe où sur l'ordinateur.



# Portfolio

Durant ce projet, nous avons eu l'occasion de paramétrer une machine virtuelle Linux afin qu'elle soit prête à l'utilisation pour des développeurs. Cela nous a appris à travailler en équipe et à nous coordonner afin de satisfaire une demande précise. De plus, nous avons pu développer nos compétences et apprendre à nous servir plus en détail de Linux et de son shell. Nous avons aussi appris à effectuer une veille d'information efficace sur le web.



```
main.go - ProjetGo - Visual Studio Code
File Edit Selection View Go Run Terminal Help

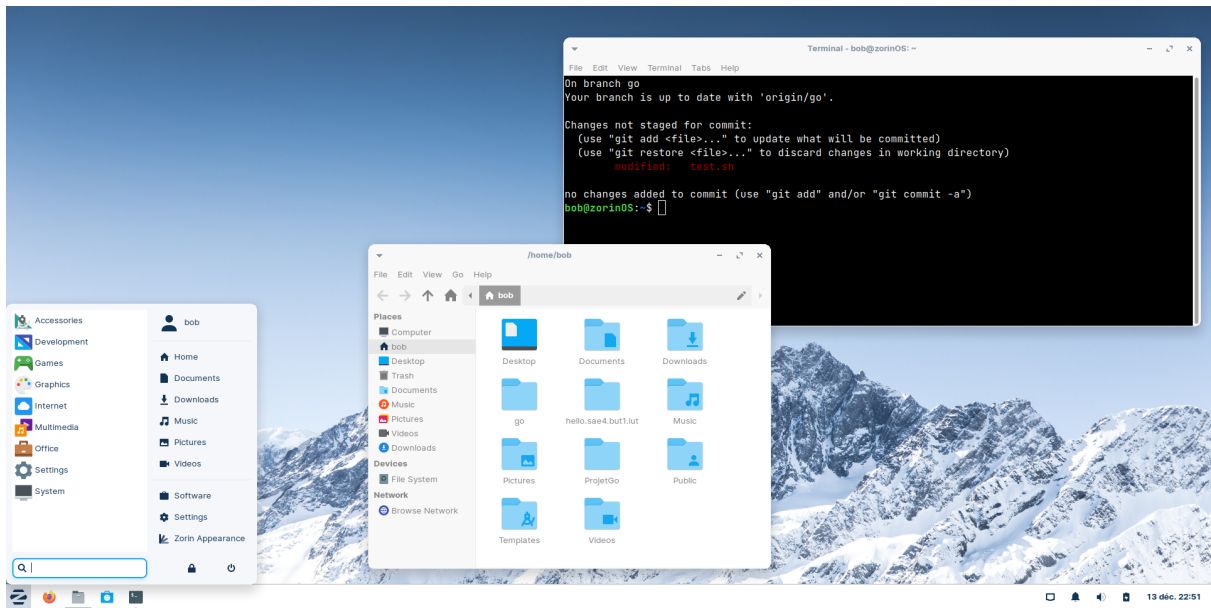
EXPLORER
PROJECTGO
└─ monProjet
   └─ main.go
      go.mod

main.go
1 package main
2
3 import "fmt"
4
5 func main() {
6     name := "world"
7     fmt.Println("hello", name)
8 }
9

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
On branch go
Your branch is up to date with 'origin/go'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   test.sh

no changes added to commit (use "git add" and/or "git commit -a")
bob@zorinos:~/ProjetGo$ go run monProjet/main.go
hello world
bob@zorinos:~/ProjetGo$
```



# Conclusion du Projet

Dans sa globalité, le projet a été mené à bien. En effet, la machine fonctionne correctement et un développeur peut dès à présent commencer à créer des projets en Go sur cette installation. Comme mentionné dans le mail, nous avons bien choisi un gestionnaire du bureau léger (XFCE), la langue système est respectée, VS Code et Git sont installés et le `.bashrc` est configuré. Cependant, nous avons rencontré quelques difficultés tout au long de la configuration. Nous avons réussi à les surmonter et à en tirer une expérience enrichissante pour la plupart. Néanmoins, certaines améliorations sont possibles.

## Difficultés surmontées et expérience

La première difficulté à laquelle nous avons fait face est celle du choix de l'OS. Nous ne connaissions pas vraiment de distributions Linux à part Ubuntu. Nous avons donc trouvé un site recensant de nombreuses distributions légères et avons choisi le plus adapté. Cela nous a incité à effectuer une veille d'informations sur Internet, développant ainsi notre capacité d'abstraction à ne conserver que l'OS qui nous paraissait le meilleur. Après, il a fallu choisir la RAM allouée à la machine virtuelle. Nous y sommes allés à taton et avons testé jusqu'à ce que la quantité allouée soit la plus optimale.

Une autre difficulté a été rencontrée un peu plus tard, au moment de la création du l'utilisateur Bob. En utilisant la commande `useradd`, nous n'avions pas fait attention qu'il fallait ajouter l'option `-m` pour que son répertoire personnel soit créé en même temps. De plus, on s'est aperçus qu'il fallait changer le shell par défaut de ce nouvel utilisateur de `sh` à `bash`. Autrement, certaines commandes ne fonctionnaient pas pour cet utilisateur.

Enfin, nous avons eu des difficultés à cloner une branche en particulier du dépôt git, nommée `go`. Après une recherche sur la manuel de la commande git, nous avons trouvé qu'il fallait ajouter l'option `-branch go` au `git clone`. Cette commande est pourtant élémentaire, et il nous est donc très bénéfique de la connaître pour nos futurs projets informatiques construits avec git.



## Amélioration possible et limites

Nous avons normalement effectué l'entièreté des configurations spécifiées dans le cahier des charges. Cependant, on aurait pu imaginer certains paramètres supplémentaires qu'on aurait pu régler sur les machines afin qu'elles soient encore plus adaptées pour une équipe de développeurs. C'est notamment le cas de la mise en place d'un serveur ssh, afin que les programmeurs puissent accéder à leur poste de travail depuis chez eux. Cependant, il nous manque énormément d'expérience dans ce domaine et on ne peut encore le faire. Cela implique la mise en réseau des machines pour qu'elles soient accessibles 24/24 heures.