Mandat

Un étudiant en informatique rêve de partir vivre en Inde quelques années et souhaite connaître quelques informations sur les salaires afin de postuler intelligemment…

Liste des fonctionnalités

1. Programme en ligne de commande (C#) pour importer les données (bdd à créer)
    a. Ne considérer que les informations avec un rating supérieur à 3
    b. Ajouter une colonne pour stocker la valeur du salaire (donné en roupie) en franc suisse
        i. Taux de change : 1 roupie équivaut à CHF 0.011.-
    c. Remplacer les **espaces** du titre de l'emploi par des **tirets**
    d. Ajouter une colonne image qui contient
        i. Si le titre de l'emploi contient android => droid.png
        ii. Si le titre de l'emploi contient python => python.png
        iii. Si le titre de l'emploi contient ios => mac.png
        iv. Si le titre de l'emploi contient sde => dev.png
        v. Et pour le reste => empty.png

2. Affichage et recherche
    a. Page générale
        i. Toutes les informations pertinentes ordonnées avec le meilleur salaire en haut (image comprise)
        ii. Afficher le salaire suisse (mettre entre parenthèses le salaire en roupie)
    b. Recherche selon le poste (sur la même page ou sur une autre)
        i. Filtrer les résultats selon le critère de recherche
        Exemple : android => affiche les emplois contenant le terme android

Annexes

Archives contenant les données : https://ici.section-inf.ch/msig-eval3

Livrable

Archive zip contenant :

- Le code source complet dans un dépôt git local
  - Un commit au moins pour la partie C#
  - Un commit au moins pour la partie WEB
  - Un commit au moins pour la documentation (voir ci-après)
- Un document de test (PDF) avec 3 scénarios décrits et effectués (exemple : https://ici.section-inf.ch/tests)
- Un vidéo qui montre
  - La page principale
  - L'affichage avec la recherche « python »

Conditions de travail

Ordinateur ETML standard, feuille de triche personnelle, conventions de codage et anciens programmes

Temps de réalisation maximum : 5 périodes

Critères d'évaluation

Objectifs et comportements observés	**Poids**
1) Fonctionnement	2
Livraison	2
Vidéo	1
GIT	1
Import->général	2
Import->rating	1
Import-->salaire	1
Import->titre	1
Import->image	1
Affichage->Accueil	2
Affichage->salaire	1
Recherche	1
pts max	84
2) Qualité du code	*1*
Structure de programmation	1
pts max	3
3) Conventions	*2*
Entête	1
Commentaires	1
Noms des éléments	1
pts max	18
4) Créativité	*1*
Innovation	2
pts max	6

Git Cheat Sheet

GIT BASICS

`git init <directory>`	Create empty Git repo in specified directory. Run with no arguments to initialize the current directory as a git repository.
`git clone <repo>`	Clone repo located at <repo> onto local machine. Original repo can be located on the local filesystem or on a remote machine via HTTP or SSH.
`git config user.name <name>`	Define author name to be used for all commits in current repo. Devs commonly use `--global` flag to set config options for current user.
`git add <directory>`	Stage all changes in <directory> for the next commit. Replace <directory> with a <file> to change a specific file.
`git commit -m "<message>"`	Commit the staged snapshot, but instead of launching a text editor, use <message> as the commit message.
`git status`	List which files are staged, unstaged, and untracked.
`git log`	Display the entire commit history using the default format. For customization see additional options.
`git diff`	Show unstaged changes between your index and working directory.

UNDOING CHANGES

`git revert <commit>`	Create new commit that undoes all of the changes made in <commit>, then apply it to the current branch.
`git reset <file>`	Remove <file> from the staging area, but leave the working directory unchanged. This unstages a file without overwriting any changes.
`git clean -n`	Shows which files would be removed from working directory. Use the `-f` flag in place of the `-n` flag to execute the clean.

REWRITING GIT HISTORY

`git commit --amend`	Replace the last commit with the staged changes and last commit combined. Use with nothing staged to edit the last commit's message.
`git rebase <base>`	Rebase the current branch onto <base>. <base> can be a commit ID, branch name, a tag, or a relative reference to HEAD.
`git reflog`	Show a log of changes to the local repository's HEAD. Add `--relative-date` flag to show date info or `--all` to show all refs.

GIT BRANCHES

`git branch`	List all of the branches in your repo. Add a <branch> argument to create a new branch with the name <branch>.
`git checkout -b <branch>`	Create and check out a new branch named <branch>. Drop the `-b` flag to checkout an existing branch.
`git merge <branch>`	Merge <branch> into the current branch.

REMOTE REPOSITORIES

`git remote add <name> <url>`	Create a new connection to a remote repo. After adding a remote, you can use <name> as a shortcut for <url> in other commands.
`git fetch <remote> <branch>`	Fetches a specific <branch>, from the repo. Leave off <branch> to fetch all remote refs.
`git pull <remote>`	Fetch the specified remote's copy of current branch and immediately merge it into the local copy.
`git push <remote> <branch>`	Push the branch to <remote>, along with necessary commits and objects. Creates named branch in the remote repo if it doesn't exist.

ATLASSIAN

Visit atlassian.com/git for more information, training, and tutorials