DIALLO Mohamed
FALL El Hadji Pathé
HOUCHAT Kamel
RALIJAONA Tomima
TILATTI Julie (chef de projet)
YANG Yi

Guide développeur

université de BORDEAUX

Année 2019 / 2020

Table des matières

I. Architecture logiciel

"*Trouve moi !*" est une application web, développée en premier lieu pour pour être utilisée sur mobile, ou un écran en orientation portrait

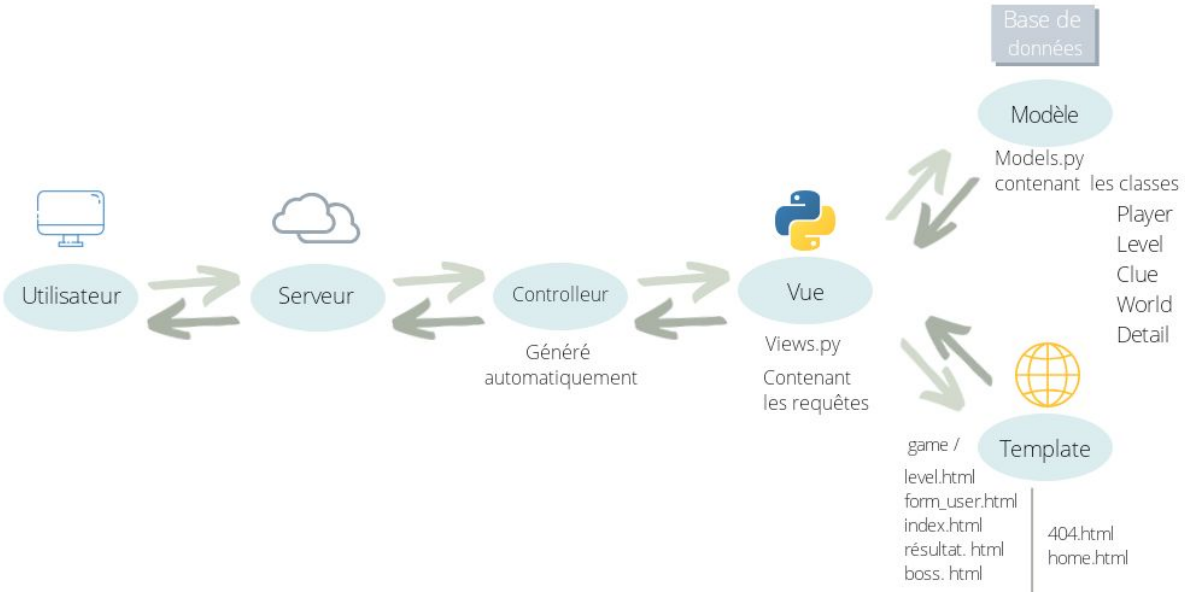
A. Architecture MVT

L'application est basée sur le framework python Django. Son système de projet et de paramètres permet une séparation nette et précise du code. Les éléments s'articulent autour du modèle MTV (Modèle Template Vue) : il s'apparente à l'architecture MVC mais son atout est qu'il génère lui-même la partie contrôleur.

Le travail du développeur est donc centré sur :

- le routage des requêtes en fonction des URLs ;
- la représentation des données dans l'application et leur gestion : Modèle ;
- l'affichage de ces données et de toutes autres informations au format html : Template ;
- le lien entre la vue qui récupère les données et génère les templates selon celles-ci : Vue.

Concrètement, lorsque l'internaute appelle une page du site, le framework se charge d'exécuter la vue correspondante, grâce aux règles de routage définies. Cette vue récupère alors les données des modèles et génère un rendu HTML à partir de Template. Une fois la page générée, l'appel fait chemin-arrière et le serveur renvoie le résultat au navigateur.



B. La base de données

1. Les tables

1- Identifiant de l'occurrence du joueur
2- Chaîne de caractères correspondant au nom du joueur
3- Nombre d'étoiles que le joueur a accumulées
4- Identifiant du niveau en cours du joueur
5- Nombre de clics réalisés pour le niveau actuel
6- Nombre total de clics réalisés au cours de la partie
7- Booléen indiquant si le premier détail du niveau en cours a été trouvé
8- Booléen indiquant si le deuxième détail du niveau en cours à été trouvé
9- Booléen indiquant si le troisième détail du niveau en cours à été trouvé
10- Booléen indiquant si le quatrième détail du niveau en cours à été trouvé
11- Booléen indiquant si le cinquième détail du niveau en cours à été trouvé
12- Identifiant de l'occurrence du monde
13- Chaîne de caractères correspondant au nom du monde
14- Identifiant de l'occurrence du niveau
15- Chaîne de caractères correspondant au nom du niveau
16- Chaîne de caractères correspondant au nom de l'image du niveau
17- Identifiant du monde auquel le niveau est rattaché
18- Identifiant de l'occurrence du détail
19- Chaîne de caractères correspondant au nom de l'image du détail

20- Nombre correspondant à la position de l'axe x du point haut gauche du détail (avec une mise à l'échelle cf. II Logique algorithmique)
21- Nombre correspondant à la position de l'axe y du point haut gauche du détail (avec une mise à l'échelle cf. II Logique algorithmique)
22- Nombre correspondant à la largeur du détail (avec une mise à l'échelle cf. II Logique algorithmique)
23- Nombre correspondant à la hauteur du détail (avec une mise à l'échelle cf. II Logique algorithmique)
24- Identifiant du niveau auquel le détail est rattaché
25- Identifiant de l'occurrence de l'indice
26- Chaîne de caractères correspondant au contenu de l'indice
27- Identifiant du détail auquel l'indice est rattaché

2. Les modèles

Chaque modèle correspond à une classe python représentant une table de la base de données et héritant de django.db.models.Model. Chaque attribut du modèle correspond à un champ de la table dont il est question.

Chaque fichier *models.py* se trouve dans un module. Pour que ces nouvelles classes soient prises en compte par l'application, il faut que le nom du module se trouve dans le fichier *settings.py* dans la parties *Installed-Apps*.

À chaque modification, il faut lancer à nouveau les migrations :

```
python3 manage.py makemigrations    // met à jour la base de données
python3 manage.py migrate           // prend en compte les modifications du fichier settings.py
```

3. L'espace administration

Django intègre un espace d'administration permettant de gérer les différentes classes de la base de données. Elle permet de modifier, ajouter ou supprimer des données.

Pour accéder à cet espace, il faut créer un "super-utilisateur" :

```
python3 manage.py createsuperuser
```

Le fichier *admin.py* permet de modifier l'espace d'administration. Pour ajouter une nouvelle table à l'affichage, il faut ajouter :

```
admin.site.register(models.[nomDeLaClasse]
```

C. Serveur local et gestion des versions

1. Présentation

Django intègre aussi un serveur web léger permettant de développer facilement. On peut donc accéder au projet en local en lançant le serveur et en accédant à http://127.0.0.1:1308/ : adresse locale sur le port 1308.

Le projet est versionné sur deux dépôts distants :

1) https://services.emi.u-bordeaux.fr/projet/git/bacchanight2019 : dépôt qui permet de versionner le projet sans être mis en ligne
2) https://github.com/bacchanight/b : dépôt qui comprend tout ce qui doit être en ligne

2. Installation

Ci-dessous, la démarche à suivre pour installer le projet :

sudo apt-get install virtualenv	// installation de virtualenv
virtualenv [Myenv]	// création d'un environnement nommé Myenv
source [Myenv]/bin/activate	// activation de l'environnement virtuel ([Myenv] correspond au chemin pour accéder à Myenv)
pip3 install django	// installation de django
git clone [dépôt 1 ou 2]	// clônage du dépôt distant 1 ou 2 (cf. partie précédente)
git remote [raccourci] [dépôt 2 ou 1]	// création d'un raccourci pour pouvoir pousser sur l'autre dépôt
python3 manage.py runserver	// lancement du serveur local
git add [fichiers]	// ajout des fichiers modifiés
git commit -m "[message]"	// création d'un commit
git push	// pousser les commits sur le dépôt distant (celui cloné)
git push [raccourci]	// pousser les commits sur le second dépôt distant
deactivate	// désactivation du l'environnement

D. Hébergement

L'application est accessible sur le web via le fournisseur de serveur PythonAnywhere. C'est un service qui permet de faire tourner des programmes python. L'application est disponible à l'adresse : http://bacchanight.pythonanywhere.com/

Notre compte est relié au dépôt : https://github.com/bacchanight/b. À chaque modification, il les récupérer avec :

```
cd bacchanight.pythonanywhere.com          // positionnement dans le bon dossier
workon bacchanight.pythonanywhere.com      // activation de l'environnement virtuel
git pull                                   // chargement des modifications
python3 manage.py collectstatic            // rechargement des données statiques (css, ...)
```

II. Logique Algorithmique

A. Connexion

La connexion est divisé en deux parties : la reprise ou le commencement d'une partie. Dans les deux cas, la logique est la même. La seule différence réside dans le fait qu'il faille un identifiant existant ou non.

1) On récupère la requête et le pseudo choisi
2) On récupère tous les joueurs
3) On compare leur nom à celui de la requête :
   - cas 1 : commencer une partie
     Dès qu'on trouve une égalité, on renvoie un erreur et on demande un nouveau pseudo
   - cas 2 : reprendre une partie
     Si on ne trouve pas d'égalité, on renvoie une erreur et on demande un nouveau pseudo
4) Si tout se déroule sans erreur, un nouveau joueur est ajouté à la base de données et il accède à la page du premier niveau

B. Gestion des clics

Le clic est la fonctionnalité principale du jeu. Elle est divisée en deux sous parties : la vérification de la position du clic et le nombre de clics.

Pour savoir si l'utilisateur a cliqué au bon endroit, voici l'algorithme utilisé :

1) Nous avons mis en place une table, qui pour chaque indice stocke les informations suivantes :
   - les positions x et y du point haut-gauche de l'image ;
   - sa longueur et sa largeur ;

- l'attribut "checked" indiquant si le détail a été trouvé ou non ;
- le contenu de l'indice et l'attribut "payedClue" ;

2) Nous avons mis en place un système de mise à l'échelle : toutes les valeurs de la table sont stockées comme des valeurs sur 100 Pour récupérer les bonnes valeurs, nous avons donc réalisé les calculs suivant :
   - position-x dans le contexte de la page : position x du détail / largeur de l'image x 100
   - Position-y dans le contexte de la page : position y du détail / hauteur de l'image x 100
   - hauteur dans le contexte de la page : hauteur du détail / hauteur de l'image x 100
   - largeur dans le contexte de la page : largeur du détail / largeur de l'image x 100
3) Les valeurs des positions x et y du clic de l'utilisateur sont ensuite comparées avec la zone créée à partir des calculs précédents.
4) Si le détail est correct, il passe à "checked", il est validé visuellement et les informations sont mises à jour dans la base de données.

Le nombre de clic quant à lui est contrôlé de la façon suivante :

- à chaque clic, la variable *click* est incrémenté ;
- si le compteur atteint la limite fixée par la variable *pauseNumber*, la variable *isClickable* passe à *false* ;
- si la variable *isClickable* vaut *false*, alors un décompte est mis en place. Lorsqu'il s'est écoulé, *isClickable* repasse à *true* et pauseNumber est remis à jour.

Scénario de la limitation des clics:

- 15 clics
- 10 secondes de pause
- 10 clics
- 20 secondes
- 5 clics
- 30 secondes
- 5 clics
- ...

C. Passage au niveau suivant

Le joueur passe au niveau suivant une fois qu'il a trouvé tous les détails. La bouton "niveau suivant" est affiché lorsque que tous les détails sont marqués comme "checked".

Lorsque le joueur clique sur ce bouton, la page est rechargée avec le niveau suivant. Les informations suivantes sont mises à jour : le nombre d'étoile, l'identifiant du niveau, le nombre de clics courant du niveau, le nombre de clics totals et les détails sont marqués comme "non-trouvés".

D. Gestion des indices

Chaque indice est relié à un détail dans la base de données. Quand l'utilisateur demande un indice, plusieurs mécanismes sont mis en places :

1) Seuls les détails qui n'ont pas encore été trouvés (qui n'ont pas l'attribut "checked") sont affichés. Tous les autres sont cachés.
2) Si l'utilisateur a déjà acheté l'indice : le détail a l'attribut payedClue, il est simplement affiché ;
3) Si l'utilisateur n'a pas assez d'étoiles, un message lui indiquant cela est affiché ;
4) Sinon, le nombre d'étoiles est mise à jour avec une requêtes AJAX et l'indice est affiché.