Vianney MORAIN
Adrien PILLET
Emmanuel CHRISTOPHE

Projet Écran d'information

Projet transverse Semestre 8
Parcours SISN

Table des matière

Introduction

Afin d'amener quelques touches de joie dans l'établissement, le CRI et Monsieur François Brucker auraient aimé avoir un écran d'affichage numérique où afficher des messages à destination des élèves et du personnel.

Ce projet consiste à développer une architecture client-serveur dans laquelle le client est un Raspberry Pi relié à un écran d'affichage et le serveur doit à terme être hébergé par le CRI.

Le cahier des charges est le suivant :

- Le Raspberry doit afficher une page web choisie à l'avance dans un navigateur en plein écran à l'allumage, et la rafraîchir régulièrement. Aucun intervention ne doit être nécessaire après la mise sous tension du Raspberry.
- Le serveur doit délivrer à une URL la page qui sera affichée sur l'écran, et disposer d'un accès administrateur afin de pouvoir changer facilement ce qui s'affiche sur cette page

Pour plus d'informations,

- Notre repo Github :
  - https://github.com/vianmora/Projet_ecrans
- Notre drive :
  - https://drive.google.com/drive/u/0/folders/1XLx0ZOIgPKmDEdcZhfp92EDsvzBj5hoM

I] Côté client : paramétrer le RaspBerry

La première étape consiste à apprendre à faire fonctionner le RaspBerry et de faire en sorte qu'une page web soit lancée à l'allumage et soit rafraîchie régulièrement. Cette page sera à terme une page web choisie par l'administrateur.

Affichage d'une page en plein écran.

Le système d'exploitation que nous avons choisi est raspbian (une version raspberry de Debian). Pour afficher une page web en plein écran, nous avons utilisé le mode kiosk de raspbian sur Chromium permettant qu'aucune barre d'outil ou champs de navigation ne soit affichée.
Ensuite, nous avons ajouté une extension en .js au navigateur Chrome permettant l'auto-refresh (code ci-dessous)

```
var numMinutes = 5;
window.setTimeout("document.location.reload();", numMinutes*60*1000);
```

Nous avons utilisé la fonction **unclutter** pour faire disparaître le curseur au bout de quelques secondes.

Contrôle à distance du raspberry

Ensuite, nous avons dû trouver un moyen de se connecter au Raspberry à distance et d'afficher l'écran à distance pour pouvoir contrôler que ce qui est affiché est bien ce que l'on souhaite même si l'écran est loin de nous. pour cela nous nous sommes connecté en **SSH** au raspberry pour le contrôler à distance et nous avons utilisé **VNC** pour afficher l'écran.

II] Côté serveur : Node.js

Page d'accueil de notre projet. Le raspberry lui se contente d'alterner entre les onglets "message du jour" et "page écran".

Structure du code

On a commencé par suivre le tutoriel du NetNinja sur Node.js pour assimiler les bases de ce langage avant de nous lancer dans la création de la page en elle même.

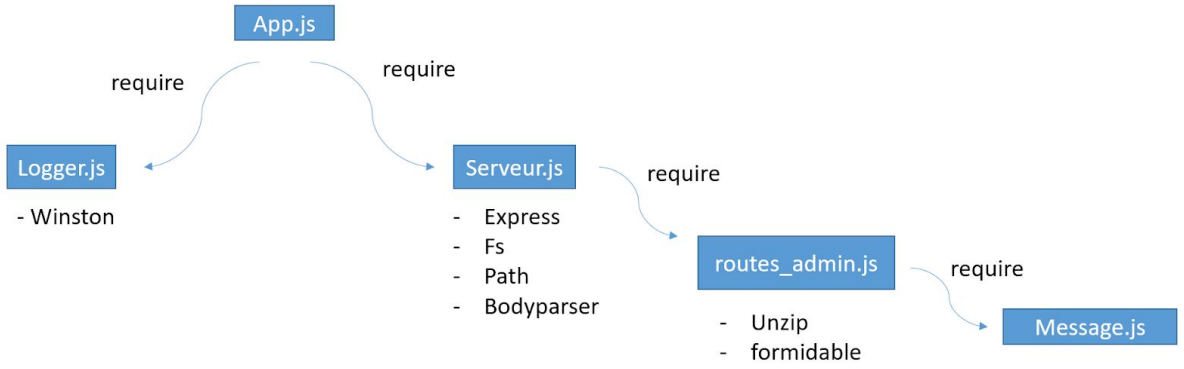
Après une première réflexion, nous allons avoir besoin des packages suivants :

- *Express*
  - pour créer un serveur simple avec Node.js
- *fs*
  - pour manipuler les fichiers
- *bodyparser*
  - pour récupérer les données envoyées de page en page
- *path*
  - pour gérer les chemins sous Node.js

En plus de ces packages de base, nous avons besoin pour le traitement administrateur :

- d'un logger : Winston
- Formidable
  - Outil pour gérer l'upload de fichiers à partir de l'interface utilisateur
- Unzip
  - Pour dézipper un fichier zip

Une fois les packages sélectionnés, nous structurons le code de la manière suivante :



Où chaque fichier javascript est importé dans le fichier au dessus de lui. En plus de ces modules, nous avons :

- un fichier package.JSON pour gérer les packages nodes installés
- un dossier views pour stocker les fichiers. ejs
- un dossier assets pour nos fichiers static utilisés par les pages html
- un dossier log pour stocker les rapports du logger
- un dossier message JSON pour stocker un historique des messages envoyés.

Enfin, nous utilisons le moteur de template ejs afin de pouvoir inclure des algorithmes de traitement dans les pages html envoyées.

Modifier le message du jour

Dans un premier temps, nous nous sommes concentrés sur la possibilité de proposer un page qui affiche un simple message. Ce message peut être modifié par un administrateur et un historique de ces messages est stocké en mémoire.

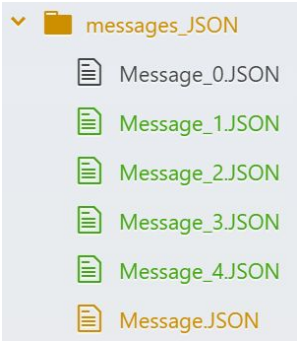
En réalité, chaque message est enregistré dans un fichier JSON numéroté. Cette structure permet de pouvoir les manipuler simplement afin d'en montrer l'historique

```ejs
<!DOCTYPE html>

<html>
  <% include partials/Head.ejs %>

  <body>

    <header>
      <h1>Projet Ecran</h1>
      <h2>Voici le message du jour</h2>
    </header>

    <section>
      <p>
        <%= message %>
      </p>
    </section>

    <% include partials/Footer.ejs %>

  </body>

</html>
```

Aperçu de la page “message du jour” et de son code html / ejs

Les fichiers JSON fonctionnent comme des dictionnaires en python où des valeurs sont associés à des mots clés. Ainsi on y stocke le numéro du message, sa date de création ainsi que le texte à afficher. Au final,Node se contente de lire le message enregistré dans Message.JSON pour l’envoyer à la page “message du jour” dans une variable “message”. Lors de l’upload d’un nouveau message, ce fichier est renommé avec son numéro pour être sauvegardé.

Pour en afficher l’historique, Node parcours donc tous les fichiers .JSON pour en extraire les informations nécessaires et les envoyer sous forme de tableau à la page “historique”.



Uploader une page

Nous avons convenu avec Monsieur Brucker qu'un bon compromis pour laisser la liberté à l'administrateur d'afficher des contenus variés sur l'écran tout en s'assurant que mettre à jour ces contenus reste une opération simple, qui peut être faite rapidement et sans connaissance particulière concernant la façon dont a été programmée le serveur était de permettre l'upload de fichiers html, éventuellement avec des éléments supplémentaires (feuilles de styles, images, polices…), le tout regroupé dans une archive .zip.

Remplacer la page écran par un simple fichier html :

Pour cette partie nous nous sommes inspiré du travail commencé par deux élèves centraliennes, Mariam Karkarashvili et Ombeline Schmidt et disponible sur le dépôt Git :
https://github.com/MariamKarkarashvili/site_utile

Comme elles, nous avons utilisé le package Formidable qui permet d'analyser des données, en particulier celles de fichiers uploadés. Ce package permet de récupérer un fichier pour ensuite pouvoir le copier/déplacer/renommer afin de l'enregistrer et le rendre utilisable. La page index.html est uploadée dans le dossier static et vient remplacer la page précédente en l'écrasant.

```
app_admin.route('/Nouvelle-page') // Pour importer une nouvelle page écran
  .get(function(req, res, next){
    res.render('a_new-screen-page');
  })
  .post(function(req, res, next){
    // Lorsqu'un fichier est uploadé:
    // On crée un nouveau recepteur formidable (tableau de fichier)
    var form = new formidable.IncomingForm();

    // L'utilisateur peut importer plusieurs fichiers d'un coup
    form.multiples = false;

    // Enregistrer les uploads dans le fichier /uploads
    form.uploadDir = path.join(__dirname, '..', '/static/uploads');

    // On renomme chaque fichier reçu avec son nom originel
    // A l'origine il a un nom du type "Upload029192..."
    form.on('file', function(field, file) {
      fs.renameSync(file.path, path.join(form.uploadDir, '/index.html'));
    });

    // informer en cas d'erreur
    form.on('error', function(err) {
      console.log('An error has occured: \n' + err);
    });

    // On décode la requète contenant les fichiers et on l'upload avec form
    form.parse(req);

    // Une fois que tous les fichiers ont été enregistrés,on envoie un eréponse au client
    form.on('end', function() {
      res.render('a_new-screen-page-success');
    });
```

Pour l'upload de fichiers zip, nous avons utilisé en plus le package Unzip qui permet de décompresser les archives.

```
    // A la fin de l'upload on envoie L'archive dans un pipe et la méthode extract d'unzip permet de déziper
    form.on('end', function() {
      var readStream = fs.createReadStream(path.join(__dirname, '..', '/static/archives/archive.zip'));
      readStream.pipe(unzip.Extract({ path: path.join(__dirname, '..', '/static/uploads')}));
      res.render('a_new-screen-page-success');
```

L'upload de fichiers html comme de fichiers zip est fonctionnel et permet de remplacer la page affichée à l'écran. Pour les fichiers zip il faut toutefois respecter certaines contraintes pour que tous les éléments s'affichent bien : nommer la page que l'on veut afficher "index.html", la feuille de style "styles.css", et placer les images et le polices d'écritures dans des dossier /images et /polices à la racine du .zip.

III] Conclusion et perspectives d'évolution

Ainsi, à l'issu de notre travail sur ce projet, nous avons réussi à avoir une interface contenant des onglets dont un onglet administrateur permettant d'importer des fichiers zippés afin d'afficher une page choisie par l'administrateur.

Voici une liste non exhaustive de pistes d'amélioration de notre projet :

- Sécuriser la page admin avec un mot de passe
- Concevoir un présentoir esthétique et sécurisé pour l'installer dans un lieu stratégique de centrale
- Mettre notre travail en production sur les serveurs du CRI
- Créer un historique des pages affichées
- L'upload des zip est soumis à encore trop de règles pour que ça puisse fonctionner à tous les coups.
- Le code peut encore être amélioré en condensant les requêtes côté serveur
- Ajouter un support physique pour le Raspberry et l'écran. Nous avons déjà découpé une boîte en bois au fablab avec http://boxes.py pour protéger le Raspberry. Pour que l'écran puisse être installé dans l'école il faudrait en plus un pied qui le maintienne ou bien un cadre qui puisse être accroché au mur.

Annexe : tutoriels suivis

Paramétrer le Raspberry :

- Le mode kiosk sur raspberry :
    - https://lofurol.fr/joomla/electronique/115-ecran-d-affichage-autonome-dashboard-avec-un-raspberry-pi

Apprendre à coder :

- Le côté serveur avec Node.js
    - https://www.youtube.com/playlist?list=PL4cUxeGkcC9gcy9lrvMJ75z9maRw4byYp
- Versionner son projet avec GitHub
    - https://openclassrooms.com/courses/gerer-son-code-avec-git-et-github