Visualisation web sur-mesure pour des codes de calcul écrits en Python

Sylvain FAURE

CNRS, Laboratoire de Mathématiques d'Orsay, Université Paris-Saclay

Motivations

Expérience personnelle (il ne s'agit pas des dates d'apparition des logiciels ou langages…) :

Préhistoire	2000	2006	2014	2016	2022
Basic Turbo Pascal Scheme Logo	Fortran90 C OCaml Gnuplot AVS Express	Python2 VTK ParaView C++ QT4	Python3 VTK C++ Cuda/OpenCL Node.JS 0.11 MEAN.JS	Python3 VTK C++17 Node.JS 0.12 MEAN.JS Three.JS, D3.JS	Python3 VTK C++22, xtensor Node.JS 16.13 Vue.JS Feathers.JS Three.JS, D3.JS

Pourquoi mettre en place une visualisation web ?

- Parce que depuis 2015 les outils existent (three.js & Co) !
- Pour proposer un service de calcul sans que l'utilisateur ait un logiciel à installer
- Plus facilement évolutive qu'une application Qt…
- Parfois plus performante

Avertissement : cette présentation a pour but de montrer via un petit projet comment faire une visualisation web sur-mesure, cependant de nombreux outils existent déjà et il faut étudier leurs fonctionnalités avant de se lancer !

Exemple de réalisation à vocation pédagogique

Modélisation du trafic routier

Modèles

MODÈLE D'ORDRE 1 | MODÈLE D'ORDRE 2

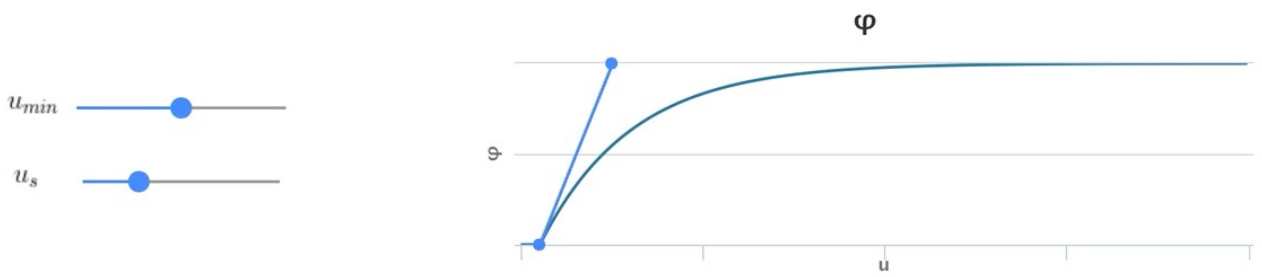
On opte pour un modèle microscopique : chaque véhicule est représenté individuellement. Soit x_i la position du véhicule i. On fait l'hypothèse que la vitesse du véhicule i ne dépend que de sa distance au véhicule directement devant lui :

$$\dot{x_i} = \varphi(x_{i+1} - x_i)$$

On propose le modèle suivant :

$$\varphi(u) = V\left[1 - exp\left(\frac{-(u - u_{min})}{u_s}\right)\right]$$

- V représente la vitesse maximale des véhicules
- u_{min} représente la distance minimale entre deux véhicules
- u_s caractérise la "nervosité" de la conduite : diminuer u_s revient à augmenter la vitesse pour une distance de sécurité donnée.



Initialisation

Nombre de véhicules 40

Modèle d'ordre 1

Code de calcul écrit en python

Principaux modules NodeJS :

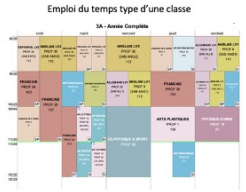
Express.js
Socket.io

Bootstrap
Angular
Highcharts
Three.js
Socket.io.client

Auteurs : C. Ramond, SF, B. Maury

Exemple de réalisation : projet CROwd COvid

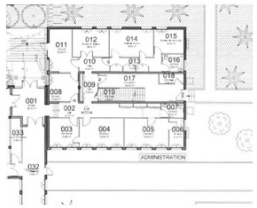
E1 Emplois du temps + listes des effectifs



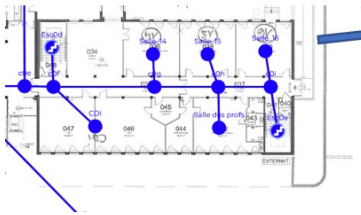
Entrées du modèle

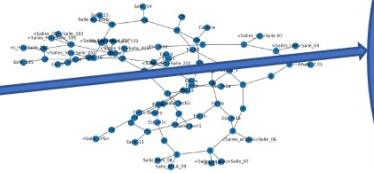
Sorties du modèle

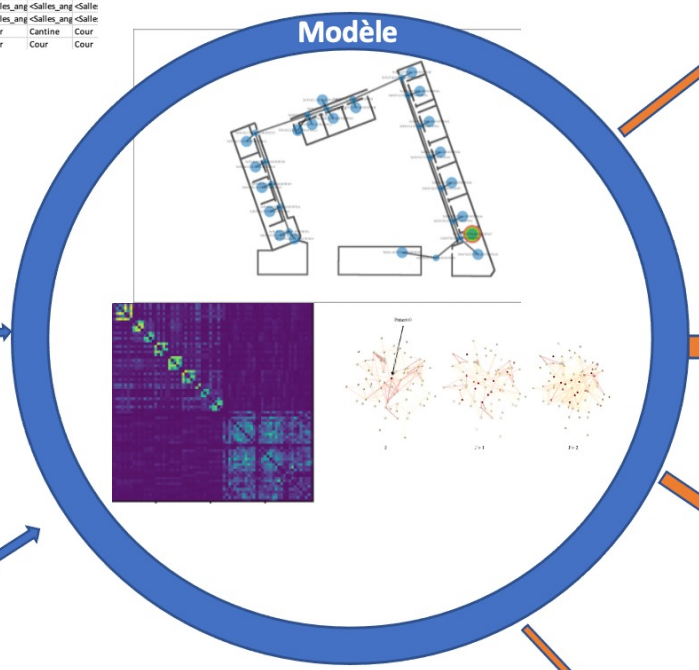
E2

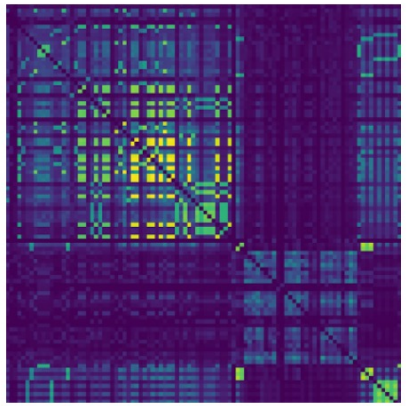


Plans







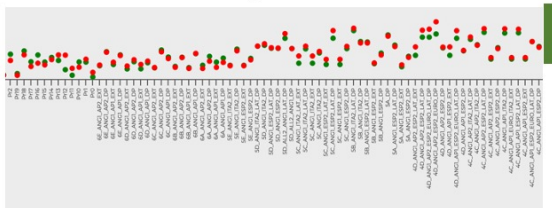


S2

Matrice de contamination

Score de risque global **S1**

Estimation des facteurs d'exposition / dangerosité entité par entité



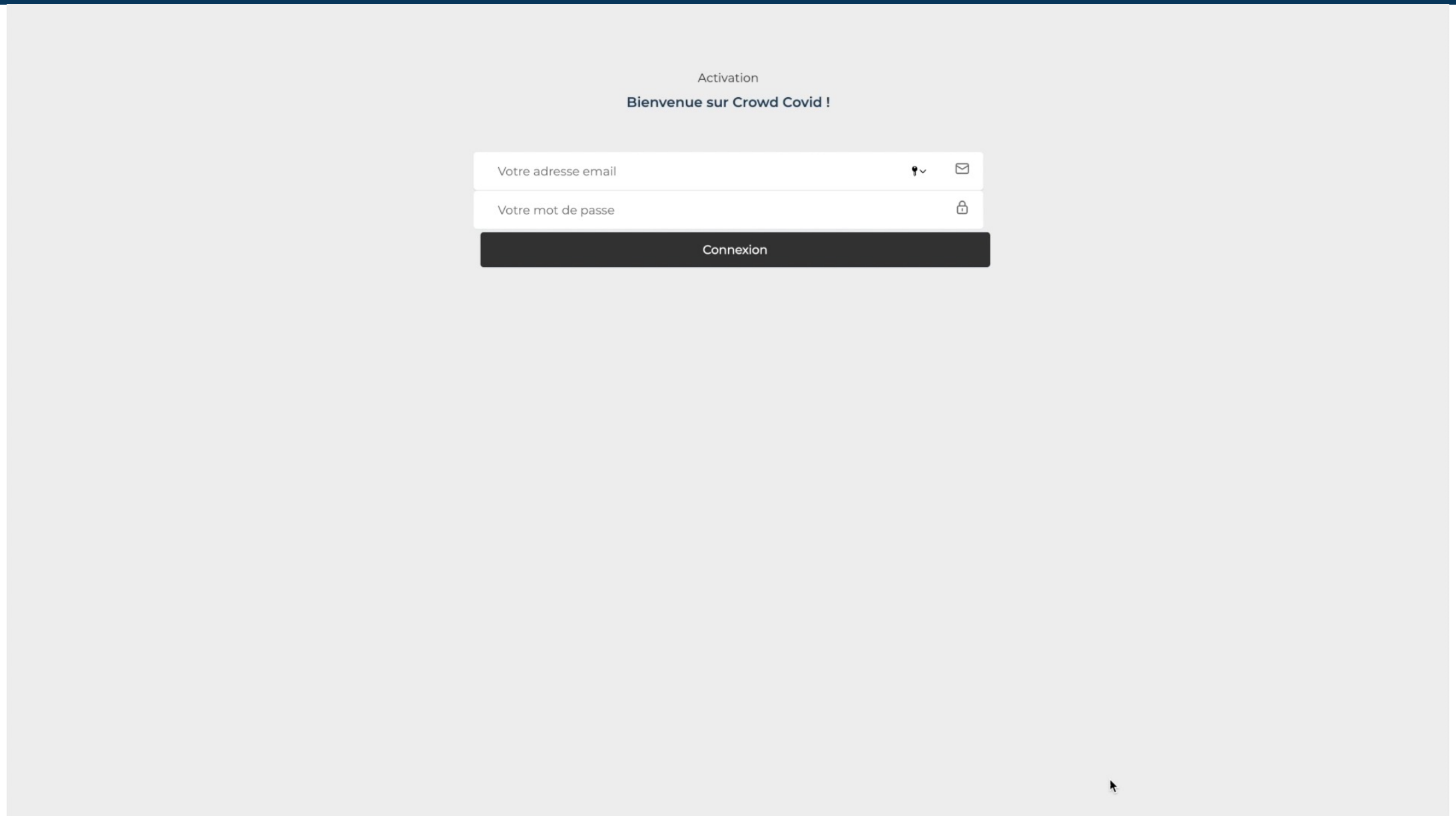
S3

« Compréhension » de l'établissement

S4

E3 Paramètres :
Période considérée, règles sanitaires en vigueur,
conditions d'utilisation des salles (ventilation, masques, ..)

Exemple de réalisation : projet CROwd COvid



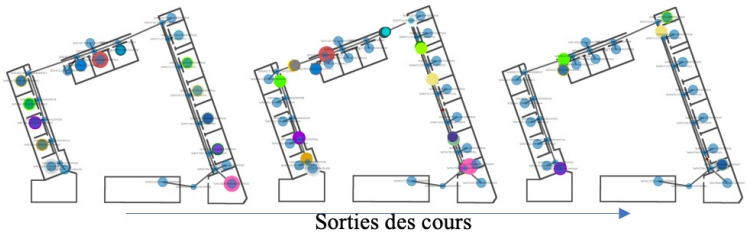
Exemple de réalisation : projet CROwd COvid

S2 + S3 → S1

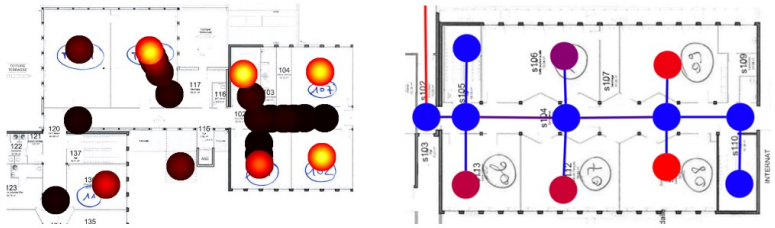
Score global : moyenne de tous les éléments,
i.e. probabilité moyenne d'être infecté

Sorties du modèle

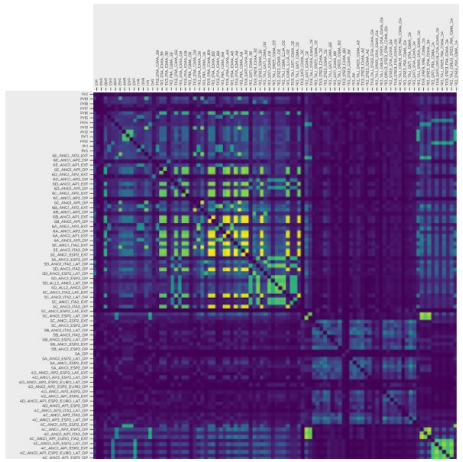
Déplacements des entités sur le graphe



Densité de contacts et temps d'occupation des locaux

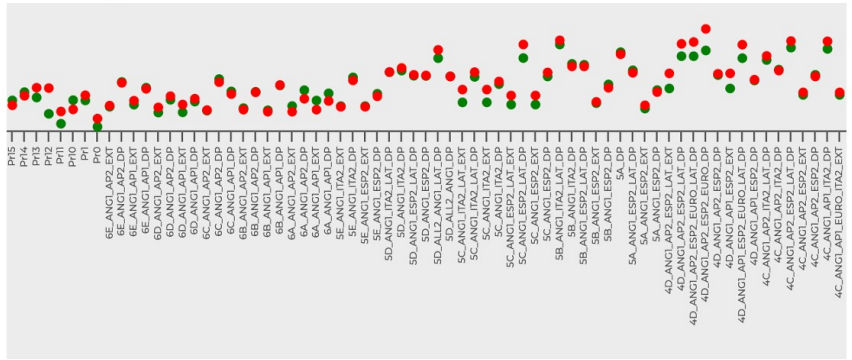


Matrice de contamination



Scores par entités

- Risque d'être exposé au virus
- Risque de transmettre la maladie si on en est atteint



Code de calcul écrit en python

Principaux modules NodeJS :

Express
Socket.io

Bootstrap
Angular
Highcharts
Socket.io.client

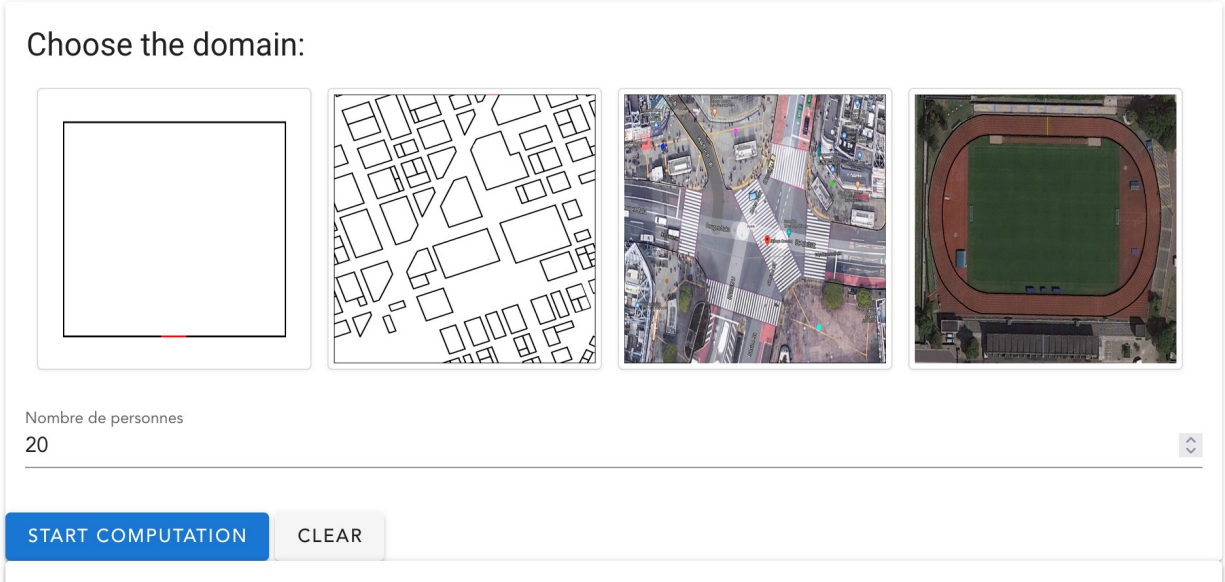
Auteurs : F. Bourdin, SF, B. Maury, G. Rouxel

Un petit projet pour débuter…

Frontend (vue.js)

Spécifications des paramètres du calcul

```
<v-col>
  <v-text-field label="Nombre de personnes"
                v-model="computation.input.N"
                type="number"
                :rules="[
                v => ((v!==null)&&(v!=='')) || 'A value is required',
                v => (v<=50) || 'Value must be less than 50',
                v => (v>0) || 'Value must be at least 1'
                ]"
                required>
  </v-text-field>
</v-col>
```



Bouton permettant de démarrer le calcul

```
<v-btn @click="start"
       color="primary"
       :disabled="!valid"> Start computation
</v-btn>
<v-btn @click="clear">Clear</v-btn>
```

```
  methods: {
    start() {
      if (this.$refs.form.validate()) {
        const bug = {
          method: 'post',
          data: {
            N: this.computation.input.N,
            background: this.imageSelected.src,
          },
          url: 'http://localhost:8082/api/computations/add',
          headers: {
            'Content-Type': 'application/json',
          },
        };
        axios(bug)
          .then((response) => {
            this.computation = response.data.computation;
          })
          .catch((e) => {
            console.log(e.response); // eslint-disable-line
          });
      }
      return true;
    },
```

Un petit projet pour débuter…

Backend (express.js)

Routes

```
// Backend : route to add a computation
router.post('/add', ComputationController.addComputation);
```

Contrôleur

```
// Add a computation
exports.addComputation = async function (req, res) {
  try {
    const input = {
      "N": req.body.N,
      "background": req.body.background,
    };
    const computation = await ComputationService.runComputation('python/micro_granular.py',input)
    computation.save()
    .then( (result) => {
      return res.status(200).json({ status: 200, computation: result, message: "Success" });
    });
  } catch (e) {
      return res.status(400).json({ status: 400, message: e.message });
  }
};
```

Modèle de données
(base de données mongoDB)

```
const ComputationSchema = new Schema({
  output: {
    id: {type: Array, default: [0,0,0,0]},
    x: {type: Array, default: [0,0,0,0]},
    y: {type: Array, default: [0,0,0,0]},
    r: {type: Array, default: [0,0,0,0]},
  },
  input:{
    background: {type: String, default: ''},
    "N": {type: Number, default: 0},
  },
  status: { type: String, default: 'inactive' },
  creationDate: { type: Date, default: Date.now },
});
```

Un petit projet pour débuter…

Backend (express.js)

Service : lancement de la ligne de commande du calcul (« child_process »)

```
exports.runComputation = async function (script, input) {
    return new Promise((resolve, reject) => {
        const cmd = 'python3';
        const json = JSON.stringify(input);
        fs.writeFile('webappjsonfile.json', json, function(err) {
          if(err) {
            logger.log(err);
          }
        });
        try{
          const args = [script, "--webappjson", JSON.stringify(input)];
          var ps = child_process.spawn(cmd, args);
          var computation = null;
          try {
            computation = new ComputationModel({
              input: input,
              //output: obj,
              status: 'start',
            });
            computation.save();
          }
          catch(comperr) {
              logger.error('Failed to create computation object in db, error = '+comperr);
          }
```

Un petit projet pour débuter…

Backend (express.js)

Service (suite) : communication avec le code python

```
        ps.stdout.on('data', function (data) {
          try {
            obj = JSON.parse(data);
            computation.output = obj;
            computation.creationDate = Date.now();
            computation.save();
          }
          catch(pyerr) {
            logger.error('--  services --> runComputation : failed to parse python results, error = '+pyerr);
          }
        } );
        ps.stderr.on('data', function (data) {
          logger.log('-- services --> runComputation (python info via stderr): '+data.toString());
        });
        ps.on('close', function (code) {
          resolve(computation);
        });
      } catch (e) {
        logger.error('-- services --> runComputation : error = '+error);
      }
    });
};
```

Un petit projet pour débuter…

Backend (python)

Le script python reçoit en argument les paramètres du calcul, le calcul démarre et c'est au processus python d'initier la communication avec l'application web.

Deux façons de faire :

- En utilisant les websockets : module « *python-socketio* » pour la partie cliente, et « *socket.io* » côté *Javascript*. Il faut juste s'assurer que les versions sont bien compatibles…
- En passant par *stdout* et *stderr* :

```
import sys
def send_err(message):
    sys.stderr.write('\n')
    sys.stderr.write(message)
    sys.stderr.write('\n')
    sys.stderr.flush()

def send_results(message):
    sys.stdout.write('\n')
    sys.stdout.write(message)
    sys.stdout.write('\n')
    sys.stdout.flush()
```

Avantage : c'est très rapide, il est possible d'envoyer de grosses données binaires grâce au format *bson* et un découpage en bloc

Inconvénients :

- Monopolise *stdout*, les logs passent par *stderr* !
- Limite de taille pour les messages *stdout*
- Pas très standard comme usage, mais depuis 6 ans cela a toujours parfaitement fonctionné…

Un petit projet pour débuter…

Remontée des résultats du calcul et visualisation

Les résultats de calculs envoyées via *stdout* sont récupérés par le **service** qui peut les stocker et/ou les renvoyer comme réponse à travers le **contrôleur**.

Au sein du service :

```
        ps.stdout.on('data', function (data) {
          try {
            obj = JSON.parse(data);
            computation.output = obj;
            computation.creationDate = Date.now();
            computation.save();
          }
          catch(pyerr) {
            logger.error('--  services --> runComputation : failed to parse python results, error = '+pyerr);
          }
        } );
```

Dans le contrôleur :

```
    const computation = await ComputationService.runComputation('python/micro_granular.py',input)
    computation.save()
    .then( (result) => {
      return res.status(200).json({ status: 200, computation: result, message: "Success" });
    });
  } catch (e) {
      return res.status(400).json({ status: 400, message: e.message });
  }
};
```

Un petit projet pour débuter…

Frontend (vue.js) et visualisation (three.js)

La fenêtre de visualisation Three.js (i.e. la vue dédiée) attend d'avoir des données pour démarrer son chargement :

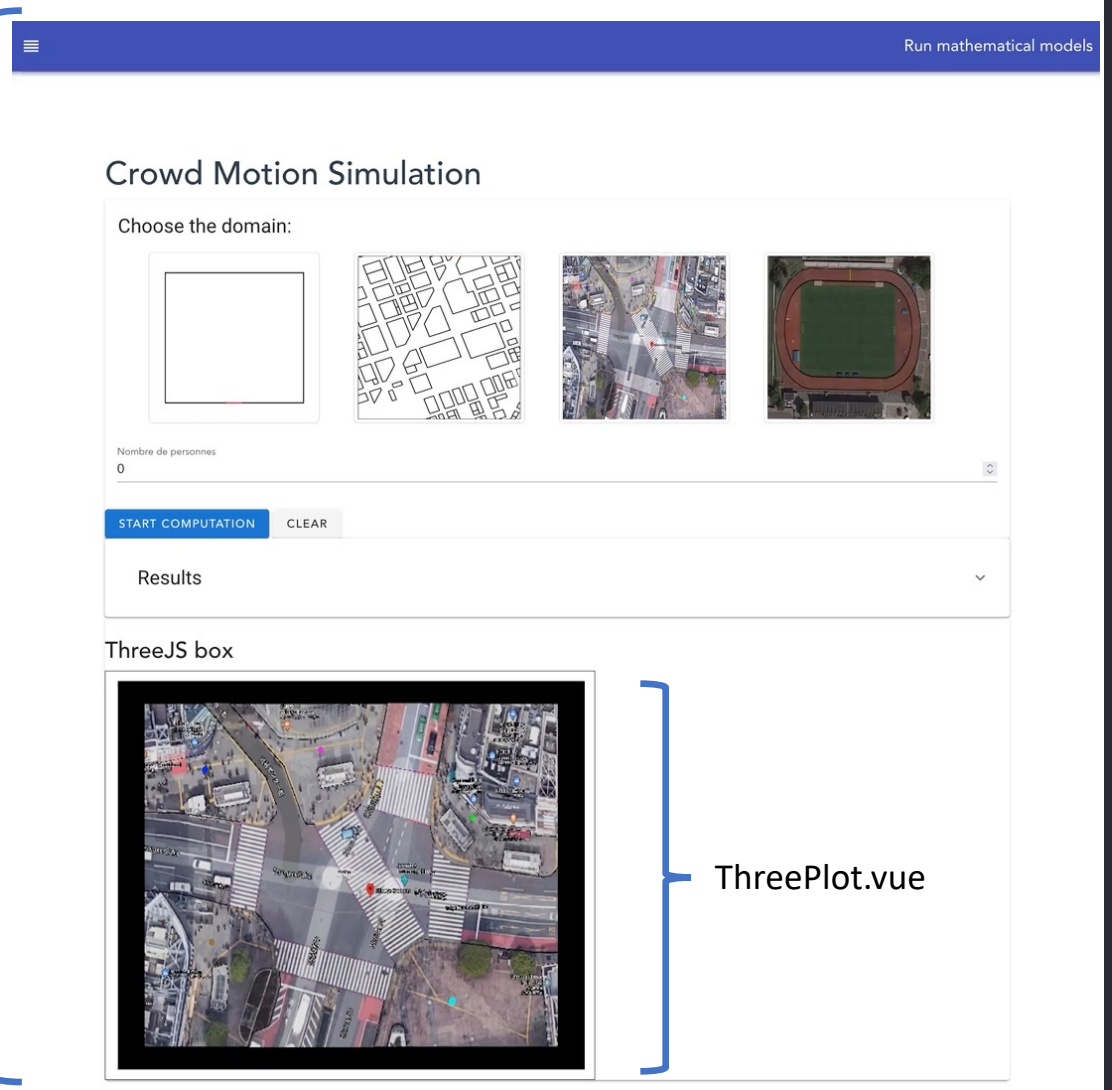
Home.vue

```
<v-card v-if="dataloaded">
  <div>
    <ThreePlot
      :current-computation=computation />
  </div>
</v-card>
```

Home.vue

```
created() {
  this.axios({
    method: 'get',
    url: 'http://localhost:8082/api/computations/last',
  })
  .then((response) => {
    if (response.data.computation) {
      this.computation = response.data.computation;
    } else {
      console.log('no last computation => default computation');
    }
    this.dataloaded = true;
  })
  .catch((error) => {
    console.log('failed, error = ', error);
  });
},
```

Home.vue



ThreePlot.vue

ThreePlot.vue

```
<template>
<div>
  <br>
  <h3>
    ThreeJS box
  </h3>
  <td style="border:1px solid #333;">
     <div class="container" ref="container"/>
  </td>

</div>
</template>

<script>
import * as THREE from 'three';

export default {
  props: {
    currentComputation: {
      type: Object,
      required: true,
    },
  },
  created() {
    // console.log('-- ThreePlot --> created');
  },
  watch: {…
  },
  static() {…
  },
  methods: {…
  },
  mounted() {…
  },
};

</script>

<style scoped>
.container {
  width: 600px;
  height: 500px;
}
</style>
```

Un petit projet pour débuter…

Frontend (vue.js) et visualisation (three.js)

La partie visualisation « pure » est standard, création de la scène, des éclairages, de la caméra et des acteurs !

```
async init() {
  const el = this.$refs.container;
  this.renderer = new THREE.WebGLRenderer();
  this.renderer.setSize(el.clientWidth - 27, el.clientHeight - 24);
  this.ctxMaxWidth = el.clientWidth - 27;
  el.appendChild(this.renderer.domElement);
  this.createScene();
  this.addBackground();
  this.createCamera();
  this.addSpotlight('#fdffab');
  this.addAmbientLight();
  this.animate();
},
```

Une fonction « watch » pour surveiller si les données (résultats du calcul) changent :

```
watch: {
  currentComputation: {
    handler() {
      this.updateBackground();
      this.animate();
    },
    deep: true,
  },
},
```

Une fonction « animate » pour déclencher le rendu lorsque les résultats ont changé :

```
animate() {
  this.updateSpheres();
  this.renderer.render(this.scene, this.camera);
},
```

Une fonction pour mettre à jour le fond (i.e. le plan)

```
async updateBackground() {
  if (this.background.name !== this.currentComputation.input.background){
    const selectedObject = this.scene.getObjectByName(this.background.name);
    this.scene.remove(selectedObject);
    this.addBackground();
  }
},
```

MERCI POUR VOTRE ATTENTION