EPSILON
ALCEN

PROJET « NOS »

NODAL ONLINE SOLVER

CCP1

Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité

Edouard PHAN CDA-01

2iSA
INSTITUT INFORMATIQUE SUD AVEYRON
Reclassement professionnel

23/09/2019
Edouard PHAN

Table des matières

1 Présentation générale

1.1 Introduction

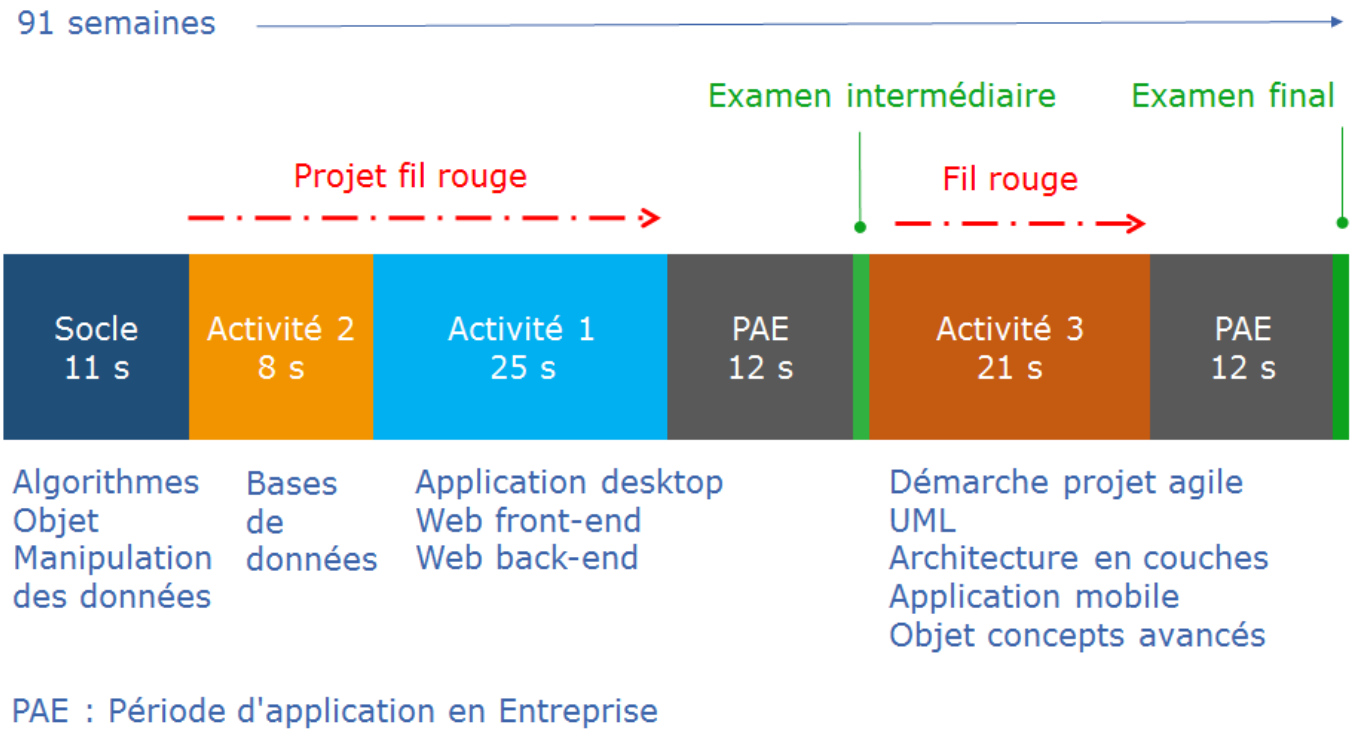
Durant ma carrière professionnelle, j'ai eu l'opportunité d'exercer dans différents domaines d'activité : dans l'enseignement de la chimie de laboratoire industrielle d'abord, puis dans la restauration en débutant en tant que commis avant de progresser jusqu'à atteindre la fonction de directeur d'entité.
Dans le cadre de ma reconversion professionnelle, encore toujours attiré par les chiffres et la résolution de problèmes, je me suis naturellement tourné vers la filière informatique. Selon moi, c'est un domaine d'avenir qui reste encore à explorer.

Pour cela, depuis janvier 2018, je poursuis une formation de **C**oncepteur **D**éveloppeur d'**A**pplication à l'**I**nstitut **I**nformatique **S**ud **A**veyron (2**ISA).**

Dans un premier temps, je vais vous présenter le titre professionnel de **C**oncepteur **D**éveloppeur d'**A**pplication qui se décline en 3 activités et 15 compétences.
(*Les compétences concernées dans ce projet inscrites en violet)

Activités	Compétences
A1 - Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité	Maquetter une application Développer une interface utilisateur de type desktop Développer des composants d'accès aux données Développer la partie front-end d'une interface utilisateur web Développer la partie back-end d'une interface utilisateur web
A2 - Concevoir et développer la persistance des données en intégrant les recommandations de sécurité	Concevoir une base de données Mettre en place une base de données Développer des composants dans le langage de la base de données
A3 - Concevoir et développer une application multicouche répartie en intégrant les recommandations de sécurité	Collaborer à la gestion d'un projet informatique et à l'organisation de l'environnement de travail Concevoir une application Développer des composants métier Construire une application organisée en couches Développer une application mobile Préparer et exécuter les plans de test d'une application Préparer et exécuter le déploiement d'une application
Compétences transverses	Utiliser l'anglais dans son activité professionnelle en conception et développement informatiques Actualiser et partager ses compétences en conception et développement

Durant la formation, nous sommes amenés à effectuer deux périodes d'application en entreprise d'une durée de 4 mois chacune.
Vous retrouverez dans ce dossier l'analyse du projet « NOS » effectué au sein de l'entreprise EPSILON du groupe ALCEN durant ma première PAE. Ci-dessous une frise permettant de vous situer dans le temps.



L'entreprise EPSILON, filiale du groupe ALCEN, est spécialisée dans la modélisation, simulation, analyse, optimisation, et conception de systèmes de hautes technologies civils et militaires, notamment dans le domaine thermique. Elle collabore avec les acteurs majeurs européens de l'espace, l'aéronautique et des systèmes embarqués.

1.2 Présentation du projet « NOS »

Le projet NOS est à l'origine une application Web destinée à représenter des modèles thermique sous forme nodale. Il a pour but de répondre à la demande des ingénieurs thermiciens de l'entreprise EPSILON dans un premier temps, pour éventuellement évoluer comme application vitrine de l'entreprise.

1.2.1 Contexte :

Dans le domaine aéronautique ou bien spatial, il est souvent nécessaire d'établir des modèles thermiques avant l'étape de fabrication, ce qui permet de rendre compte des contraintes dues aux différents matériaux employés et à l'environnement thermique. L'entreprise EPSILON a réalisé une application permettant de concevoir et visualiser des modèles thermique simplifiés sous forme nodale. Ces modèles sont transmis à un solveur qui calcule et retourne les résultats thermiques de la simulation. Ces modèles simplifiés engendrent des approximations mais leur permet de filtrer rapidement les modèles en fonction des contraintes.

La problématique exposée est la gestion simplifiée des modèles thermiques sous forme nodale, afin de représenter pour chaque nœud un point thermique muni de ses propriétés et les liaisons représentant les différents types d'échanges thermiques. Ces données sont ensuite envoyées au format Json à un Mailer qui les convertit au format TXT pour les envoyer ensuite au *SOLVER*, ce qui représente la partie calcul thermique. Les résultats de ces calculs sont envoyés au format TXT au Mailer qui convertit ces données au format XLSX pour les retourner au client. Comme vous avez pu le constater, la gestion des données est plutôt fastidieuse.

Il nous a été demandé à mon binôme et moi-même de :

EPSILON

- Modifier la partie back-end
- Supprimer le *MAILER* pour le remplacer par une API
- Améliorer les fonctionnalités de l'application notamment l'interface (Evolution vers Bootstrap 4)
- Uniformiser les langages utilisés (en utilisant TypeScript le plus possible)

2ISA

- Livrer pour le mercredi 25 septembre 2019 un dossier technique comprenant les éléments suivants :
    - Présentation et reformulation du besoin
    - Résumé du projet en anglais d'une longueur d'environ 20 lignes soit 200 à 250 mots
    - Architecture technique mise en œuvre
    - Suivi de notre activité et de ce qui reste à faire.
- La présentation orale, avec un support Powerpoint comprenant les éléments suivants :
    - Présentation au jury du candidat commençant par un résumé en anglais de son projet.
    - Rappel des objectifs de ce travail et des compétences visées
    - Situation du contexte de la réalisation, reformulation des contours du projet et des objectifs fonctionnels
    - Présentation de ma démarche en mettant en évidence les principales étapes de mon travail
    - Présentation des résultats en argumentant mes choix, les difficultés rencontrées et les solutions apportées.
    - Présentation d'une conclusion, en faisant le point sur mon état d'avancement et en dressant un bilan du projet

Ces évolutions font suite une réunion interne à EPSILON. Vous retrouverez le compte rendu et les perspectives d'évolution.

NOS

Utilisation:

- Outils de prédimensionnement
  => calcul simple et rapide

Atouts	A améliorer
- Visualisation du modèle - Interface - Simplicité d'utilisation	- Ergonomie - Fonctionnement lourd par mail - Pas de dépendance à la température

Développement par ordre de priorités

1. Architecture client - API - solveur
2. Amélioration de l'ergonomie/ interface
3. Dépendance à la température
4. Résultats visuels (post-traitement dans l'interface)

- Liste matériaux
- Formules standards de conduction/convection/radiation/VF
- Etudes paramétriques et post-traitement adapté

Améliorations envisageable

- Génération de rapport visuel de modèle systema
- Visualisation de bilan de flux
- Import de CAO
- Calcul de facteurs de vue automatisé
- Mode standalone / offline
- Noeuds "intelligents" = modèles systèmes disponibles en tant que noeuds
- Machine learning couplé aux études paramétriques pour trouver un optimum
- Gestion de versions de modèles

Outil de prédimensionnement

- Faisable en systema / Thermisol, mais un peu lourd => écriture en fichier texte

Atouts

- Visuel*
- Interface*
- Ergonomie*
- Dépendance à la température*
- Résultats visuels*
- Formules standards de conduction/convection/radiation/VF**
- Liste de matériaux**
- Générateur de noeuds / formes standards
- Ne pas passer par les mails*
- Etudes paramétriques et post-traitement adapté**
- Machine learning couplé aux études paramétriques pour trouver un optimum
- Gestion de versions de modèles

Autres idées de fonctions

- Génération de rapport visuel de modèle systema
- Visualisation de bilan de flux
- Import de CAO
- Calcul de facteurs de vue automatisé
- Mode standalone / offline
- Noeuds "intelligents" = modèles systèmes disponibles en tant que noeuds

1.3 Cahier des charges

1.3.1 API

Un serveur NodeJS écrit en TypeScript doit servir d'intermédiaire entre le solveur et le client. Les routes seront écrites avec le Framework Express

Les fonctionnalités de l'API doivent être couvertes avec des tests unitaires écrits avec le Framework Jest.

L'API doit répondre aux exigences suivantes:

- Méthode de conversion XLSX => JSON
  - la méthode doit prendre en entrée un fichier Excel
  - la méthode doit retourner une erreur si le fichier ne contient pas toutes les feuilles attendues
  - la méthode renvoie une erreur si chaque feuille ne contient pas toute les colonnes attendues
  - la méthode est testée par un test unitaire utilisant les fichiers étalons
- Méthode de conversion JSON => XLSX
  - la méthode prend en entrée un objet JSON défini dans l'annexe 2
  - la méthode doit retourner un fichier Excel défini comme dans l'annexe 1
  - pour générer le fichier Excel, la méthode utilise la librairie XLSX
- Route d'import de données
  - chemin "model/import"
  - méthode POST contenant un fichier Excel dans le body
  - paramétrer "*multer*" pour utiliser un Stream et ne pas écrire le fichier sur le server
  - contrôle si le fichier Excel est valide (contient les feuilles et les colonnes)
  - appel de la méthode de conversion XLSX2JSON écrite précédemment
  - retourne une réponse contenant le modèle au format JSON
  - la route est testée par un test unitaire utilisant les fichiers étalons
- Route d'export de données
  - chemin "model/export"
  - méthode POST contenant le modèle au format JSON
  - appel de la méthode de conversion JSON2XLSX écrite précédemment
  - retourne une réponse contenant un fichier Excel
  - la route est testée par un test unitaire utilisant les fichiers étalons
- Route exécution de calcul
  - chemin "model/compute"
  - méthode POST contenant le modèle au format JSON
  - écriture du modèle dans un fichier sur le serveur
  - appel du solveur
  - lecture du fichier de résultats écrit par le solveur
  - retourne une réponse contenant le modèle au format JSON
  - la route est testée par un test unitaire utilisant les fichiers étalons

- Gestion des erreurs
  - un middleware express pour la gestion des erreurs de l'API
  - en cas d'erreur l'api renvoie un objet ayant les propriétés suivantes:
    - Statuts (valeur par défaut: 500)
    - Détail (valeur par défaut: "*internal server error*")
    - Message

1.3.2 Client

Le client est écrit dans le Framework Angular et utilise les librairies ngxBootstrap et D3js.

- Exécution d'un calcul
  - modifier le model service pour appeler la route "/model/compute"
  - générer la requête http avec la méthode post contenant le model
  - mettre à jour le model avec la réponse de la requête
  - modifier le menu pour appeler cette nouvelle route dans le service
- Import d'un fichier
  - modifier le menu et supprimer le jQuery
  - modifier le menu pour appeler cette nouvelle route dans le service
  - modifier le model service pour appeler la route "/model/import"
  - générer la requête http avec la méthode post contenant le fichier Excel
  - mettre à jour le model avec la réponse de la requête
- Export d'un fichier
  - modifier le model service pour appeler la route "/model/export"
  - générer la requête http avec la méthode post contenant le model
  - télécharger le fichier Excel contenu dans la réponse
  - modifier le menu pour appeler cette nouvelle route dans le service
- Vue résultats
  - créer une route "results" dans le module de routing
  - créer un composant "resultList" qui contient un tableau contenant les résultats des calculs
  - ajouter un lien dans le menu pour afficher la vue "results" (le lien doit être désactivé si le modèle ne contient pas de résultat)
  - à la réception de la réponse du la route "compute", naviguer vers la vue "results"
- Loader
  - créer un httpIntercetor(voir doc Angular) pour intercepter les requêtes
  - créer un composant loader contenant une fenêtre modale et un GIF

1.4 Environnement et outils techniques:

Voici les différents outils utilisés pour modifier cette application web :

Web Storm Environnement de développement Intégré

Langage TypeScript

Plateforme logicielle

Plateforme de développement

Framework Express

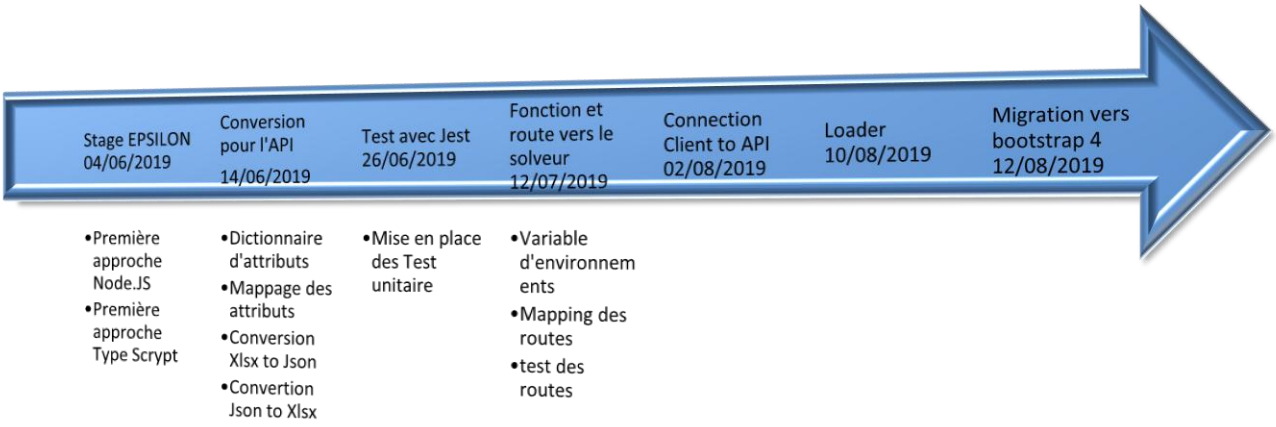
Système de Version distribué

Outils Test d'API

Le HTML est un langage dont la fonction est gérer et organiser le contenu d'une page web. C'est un langage de description de données, et non un langage de programmation.

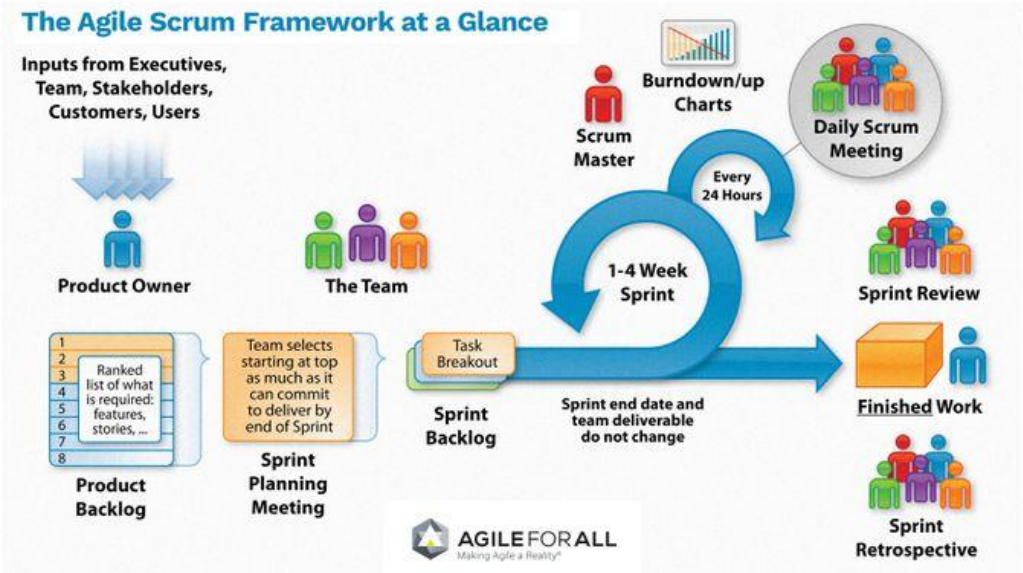
Le rôle du CSS est de gérer l'apparence de la page web (agencement, positionnement, décoration, couleurs, taille du texte…). Ce langage est le complément du langage HTML pour obtenir une page web avec du style. Le navigateur parcourt le document HTML. Lorsqu'il rencontre une balise, il demande à la CSS de quelle manière il doit l'afficher.

1.5 Chronologie du projet :



1.6 Démarche et réalisation

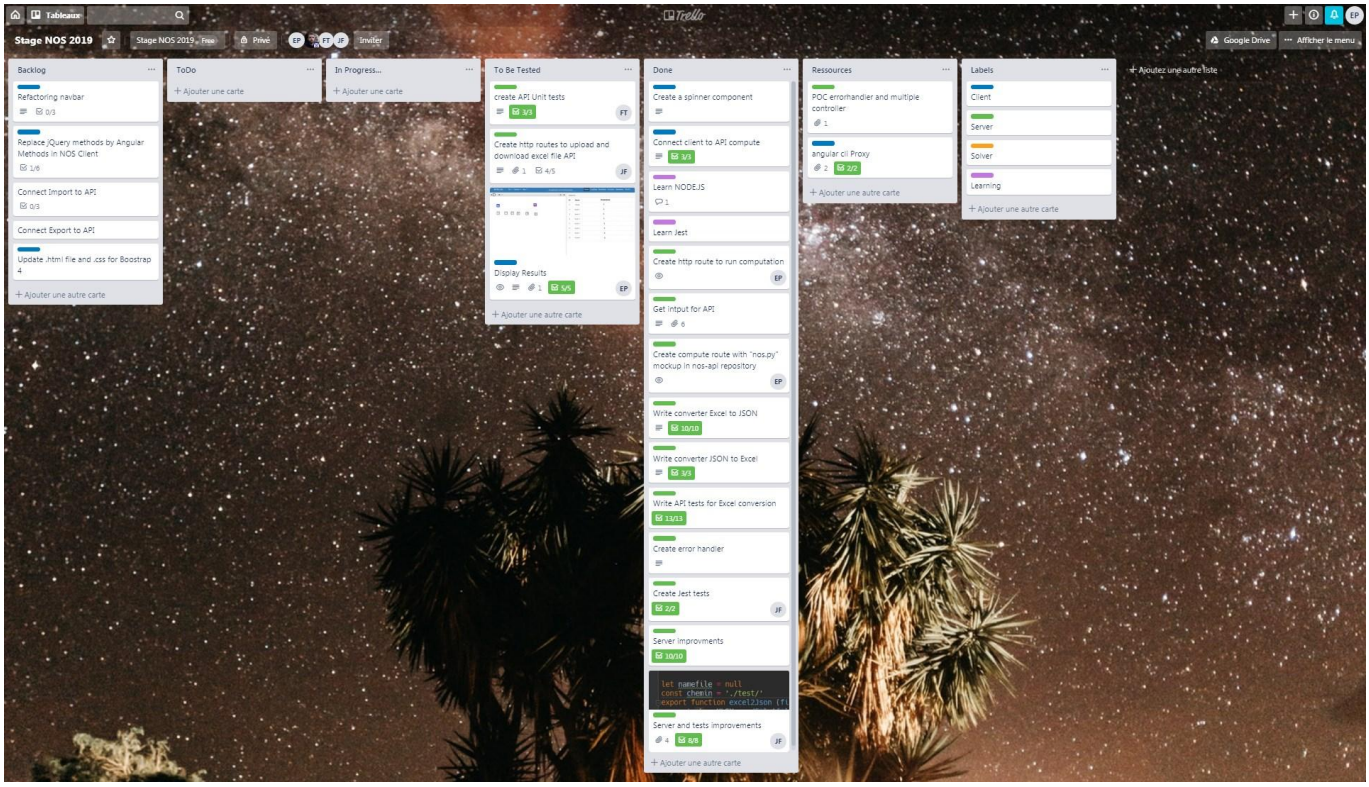
Le projet est conduit par le Product Owner Francis TOSOLONI et notre maître de stage Loïc VIENNOIS.
La démarche proposée pour mettre en œuvre les différents changements est le FRAMEWORK AGILE SCRUM auquel j'ai été initié au sein de l'entreprise ARKADIN.



Pour démarrer le projet, Francis TOSOLONI et Loïc VIENNOIS se sont concertés afin de définir un premier jet du BACKLOG. Un compte-rendu des tâches effectuées via SLACK est réalisé tous les jours afin de mesurer de l'état d'avancement du projet.
Il est ainsi possible de fournir au Product Owner une démo ou réalisation, afin de pouvoir ajuster la « *vision* » aux besoins du client.

1.7 BACKLOG

Le tableau d'avancement du back log est réalisé à l'aide de Trello



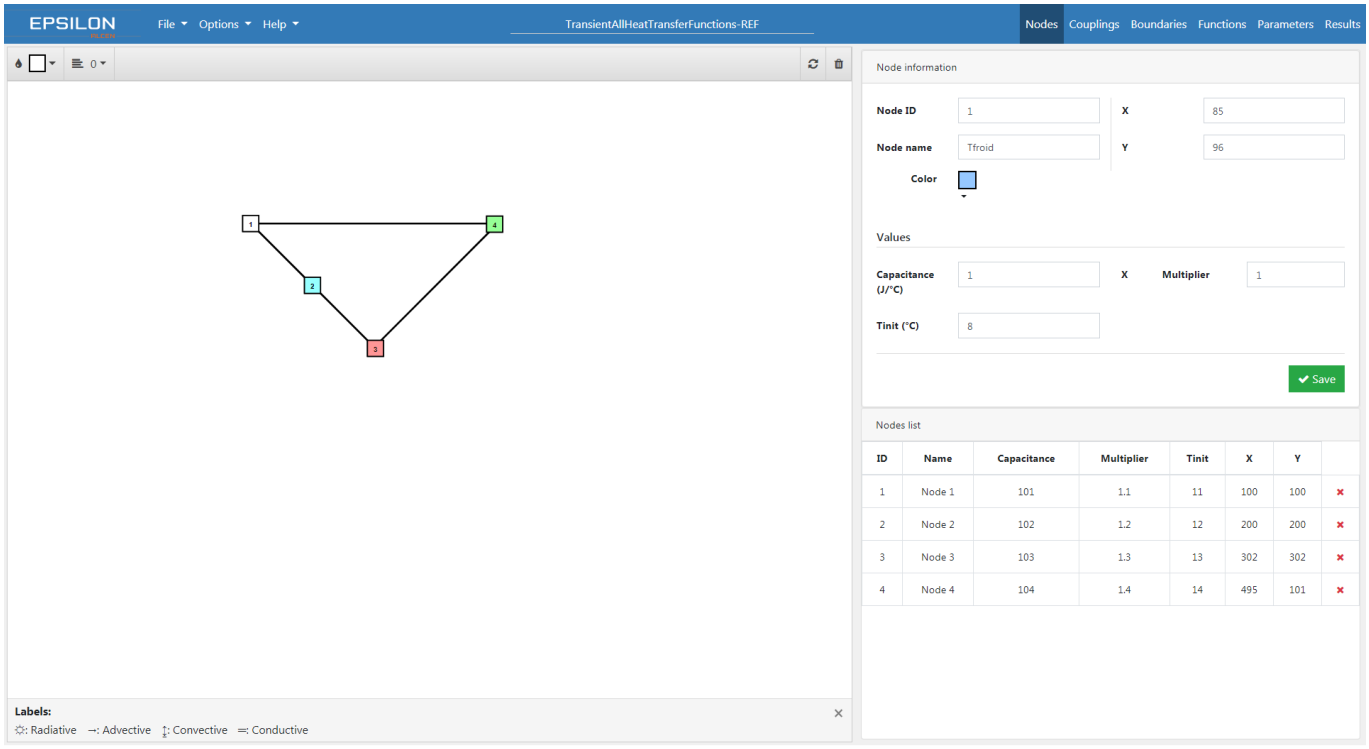
2 Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité CCP1

Compétences évaluées dans ce projet :

- Développer la partie back-end d'une interface utilisateur web
- Développer des composants d'accès aux données
- Développer la partie front-end d'une interface utilisateur web

2.1 Présentation de l'application NOS

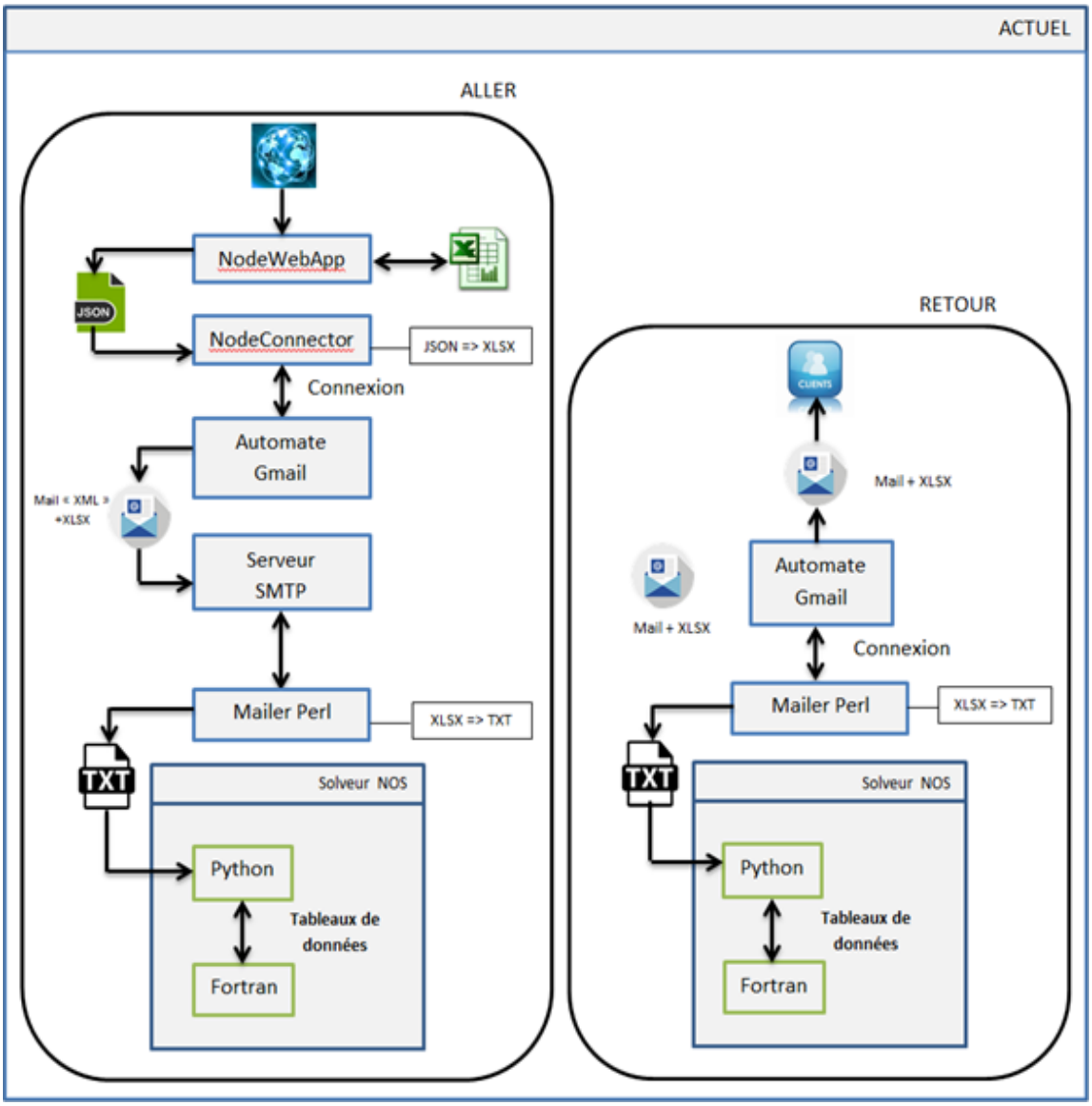
Dans cette partie, je détaille le plan mis en œuvre et les fonctionnalités apportées à l'application.



Pour ce faire, j'ai commencé par la reprise et la modification de la partie API « BACK-END », en démarrant par le module de conversion de Json vers Excel puis d'Excel vers Json. J'ai ensuite créé l'API destinée à communiquer entre le client et le solveur. Une fois l'API terminée, les tests et gestions erreurs ont été établis.

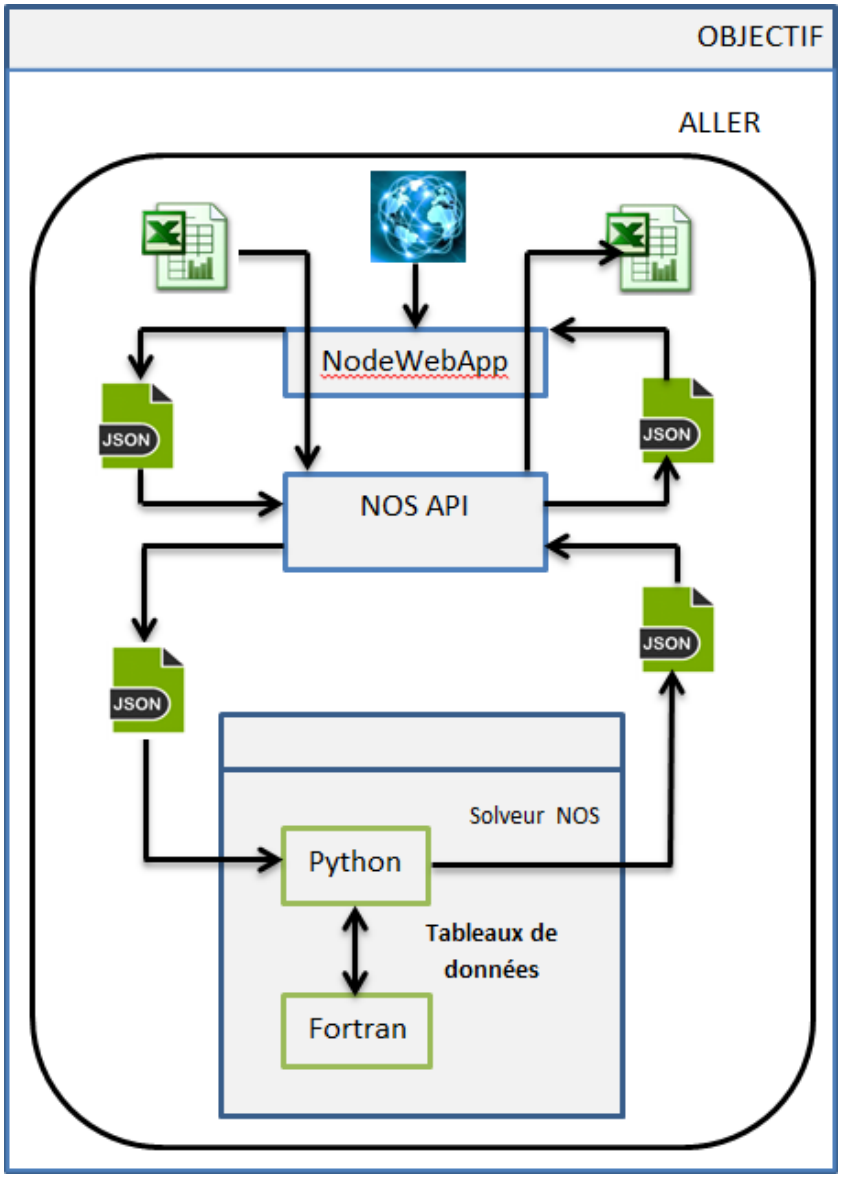
J'ai ensuite pu reprendre la partie CLIENT pour adapter les routes ainsi qu'effectuer la migration vers une version plus récente de BOOTSTRAP (la version 4) tout en conservant un affichage similaire aux précédentes versions.

2.1.1 Architecture



2.1.1.1 L'existant :

EPSILON possède une solution logicielle capable de créer de manière interactive un modèle thermique nodale et est associé à un solveur thermique nodal.
Un modèle nodal peut être créé depuis l'interface ou l'utilisateur peut uploader un fichier XLSX respectant un Template défini. De manière sous-jacente, le fichier XLSX est transféré par mail à un mailer PERL. Celui-ci convertit les données au format TXT pour les donner en entrée du solveur NOS. Le solveur NOS est écrit en PYTHON et FORTAN. Les résultats générés par le solveur au format TXT sont réinjectés dans le fichier XLSX (dans un nouvel onglet) et le tout est envoyé par le mailer PERL dans la boite mail de l'utilisateur. Le code client de l'application est écrit en TypeScript avec le Framework Angular, et utilise la librairie Bootstrap 3 pour la mise en forme HTML/CSS. Il n'est actuellement pas possible de récupérer les résultats directement dans l'interface.



2.1.1.2 L'objectif :

L'objectif des travaux de développement est de simplifier l'application aussi bien du point de vue technique que fonctionnel. Pour l'utilisateur toutes les actions (édition, calcul, lecture des résultats) devront pouvoir se faire depuis l'interface Web. Côté technique, le nombre de couches applicatives doit être réduit de 5 à 3. Une API remplacera les rôles du serveur mail, du "nodeConnector" et du mailer Perl. Cela permettra également d'uniformiser les langages utilisés (en utilisant TypeScript le plus possible) et de n'utiliser plus que 2 formats de données (XLSX et JSON).

L'API sera écrite avec le moteur NodeJS et la librairie Express pour la gestion des routes. L'API se chargera de l'import et de l'export des données (conversion XLSX <->JSON) et du lancement du solveur. L'API renverra les résultats générés par le solveur directement à l'interface.

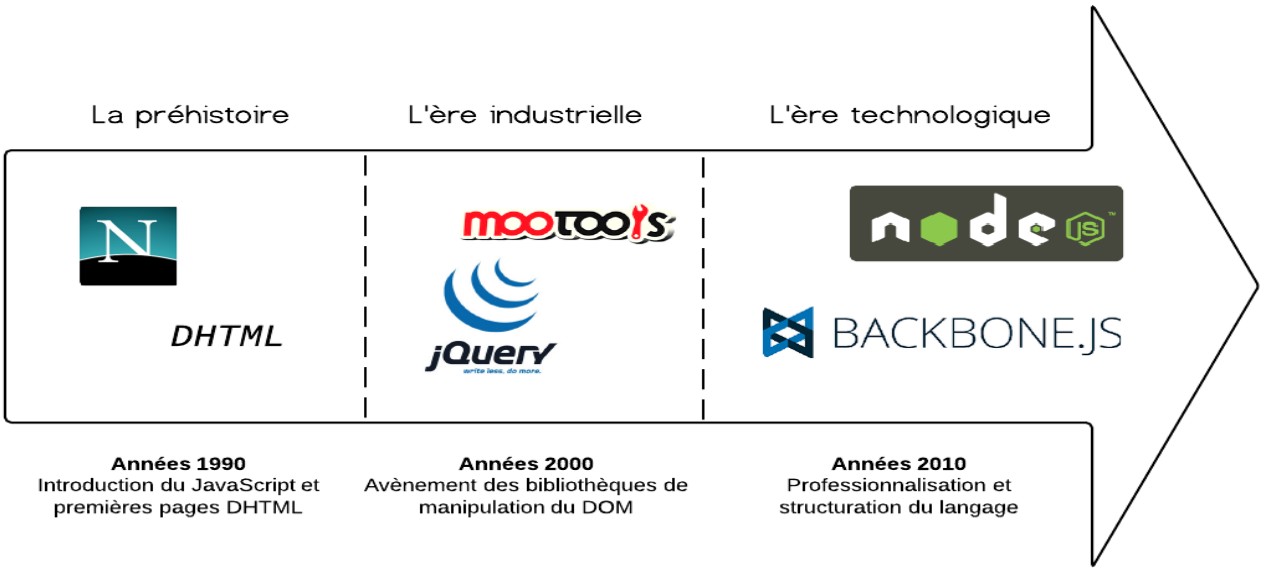
Le client Angular doit être modifié pour utiliser cette nouvelle API (import/export de modèle et récupération des résultats). Une première version d'affichage des données sera ajoutée à l'interface.

2.2 Développer la partie back-end d'une interface utilisateur web, NOS Partie API

2.2.1 Reprise et modification de la partie API

Suite à la suppression de la partie MAILER destinée à transmettre les données XLSX au solveur, nous avons commencé par la mise en place des méthodes conçues pour l'API permettant la conversion des fichiers XLSX en JSON et inversement.

Pour ce faire nous avons employé Node.JS selon la norme ECMA Script à travers le langage TypeScript. Avant de vous parler de son utilisation, voici un petit historique de JavaScript, un langage longtemps ignoré ou déprécié par les développeurs.



NodeJS est un moteur JavaScript permettant l'utilisation de ce langage sur le serveur, donc en dehors du navigateur. NodeJS bénéficie de la puissance de JavaScript pour proposer une toute nouvelle façon de développer des sites web dynamiques.

NodeJS est basé sur le concept de boucle d'événements. Par conséquent, c'est toute la façon d'écrire des applications web qui change.
Et c'est de là que NodeJS tire toute sa puissance et sa rapidité.

Étant donné que JavaScript est un langage conçu autour de la notion d'évènement, NodeJS a pu mettre en place une architecture de code entièrement non-bloquante (asynchrone), notamment grâce aux fonctions « CALLBACK ».
En JavaScript, on peut tout à fait envoyer une fonction en paramètre d'une autre fonction. Cela signifie : " Exécute cette fonction quand l'évènement est terminé ". Ainsi, dans l'exemple d'un téléchargement, l'utilisateur n'est pas obligé d'attendre que le premier téléchargement soit terminé pour lancer le deuxième.

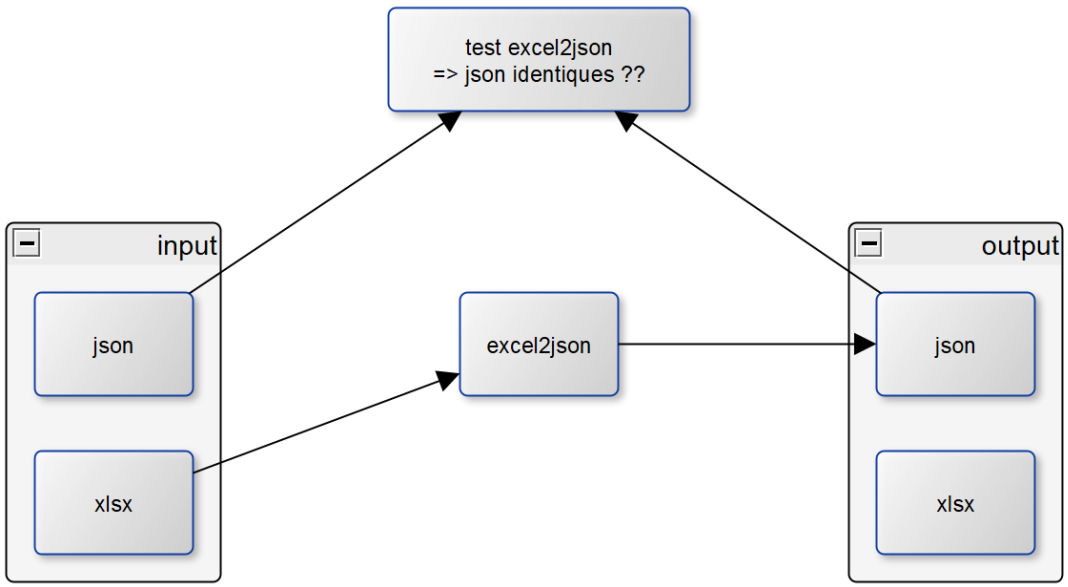
1 https://nodejs.org

2.2.2 Méthode nécessaire pour la conversion xlsx to Json

Pour commencer le projet, j'ai tout d'abord récupéré le repository Git de la dernière version de NOS depuis GitLab[2].

Git est un système de version distribuée me permettant de collaborer avec Jean-Yves FONTENIL un camarade de promotion. Il est ainsi possible de travailler en même temps sur le projet, mais de préférence sur des fichiers distincts pour éviter les conflits.

Après avoir appréhendé le NodeJS et le TypeScript durant les 2 premières semaines du projet, j'ai commencé par la création de fonctions destinées à la conversion des fichiers **XLSX** avec un modèle en format **JSON** avec la librairie npm « xlsx » (voir annexe « Conversion ») . Suite à la réalisation du module chargé de la conversion Excel vers Json et Json vers Excel, il a été nécessaire de convertir les attributs par le biais d'un dictionnaire, afin de faire correspondre les attributs Excel avec les attributs JSON utilisés par l'interface et le solveur (voir annexe « Dictionnary »).



J'ai ensuite conçu des fonctions dédiées aux tests afin de vérifier le fichier XLSX avant traitement (vérification du nom des onglets, de l'attribut des colonnes…) Puis, à l'aide de Jest, un Framework permettant l'écriture et l'automatisation des tests, j'ai établi les « Tests Unitaires » afin d'améliorer la robustesse du code.

2 https://about.gitlab.com/

2.2.3 Gestion des erreurs

Pour la gestion des erreurs j'ai créé une fonction *errorHandler* destinée à récupérer les erreurs et en afficher le contenu, à savoir le « status » et « error ».

```
export function errorHandler(err: any, req: Request, res: Response, next: NextFunction) {
  let error: ErrorMessage
  if (!(err instanceof ErrorMessage)) {
    error = ErrorMessage.create("", {error: err})
  } else {
    error = err as ErrorMessage
  }
  console.log(error.error)
  res.status(error.status);
  res.send(error.toHttpResponse())
}
```

2.2.4 Tests Unitaires

J'ai ensuite établie des tests unitaires avec le Framework Jest[3] afin de vérifier l'intégrité des données à partir des fichiers spec.ts et comparer les fichiers XLSX et JSON d'entrées avec les sorties. Pour cette partie, nous avons utilisé l'outil « *isequal* » de la librairie npm « lodash », qui permet de comparer deux objets.

Après la gestion des erreurs, le Product Owner Francis TOSOLINI m'a demandé de rendre les « Response, res.status » plus générique par le biais d'une classe, et ainsi faciliter la gestion des erreurs et leur retour.

Pour cela, j'ai dû maitriser les principes des **middlewares** d'Express[4] permettant d'effectuer un traitement avant celui défini par les routes. Vous retrouverez ci-dessous la classe implémentée pour les erreurs destinées au « next () ».

```
export class ErrorMessage implements Error {

  name: string = ""
  status: number
  message: any
  description: string

  constructor(message: any, description?: string, status?: number) {
    this.message = message
    this.status = status || 500
    this.description = description || "Internal server error"}}
```

3 https://jestjs.io/
4 https://expressjs.com

2.2.5 Route exécution de calcul

Faisant suite au modèle de conversion, le chef de projet m'a confié la réalisation d'une fonction destinée à exécuter un script Python permettant de simuler le comportement du solveur.

Pour cela, j'ai dû appréhender

- Les variables d'environnements permettant de définir des chemins,

  - `RUN_FOLDER="C:/Users/EP/NOS_EPSILON/nos-api/run"`
  - `SOLVER_PATH="C:/Users/EP/NOS_EPSILON/nos-api/solver/nos.py"`
  - `PYTHON_PATH="C:\Users\EP\Miniconda3\python.exe"`

- L'utilisation « d' **Express** » destiné aux « mapping des routes méthodes » à appeler.

  ```
  export class ThermalModelRoutes {
      public thermalModelController: ThermalModelController = new
  ThermalModelController()
      public computeController: ComputeController = new ComputeController()
      public routes(app: any): void {
          app.get('/thermalmodel', this.thermalModelController.getThermalModel)
          app.get('/thermalmodel/import',
  this.thermalModelController.importExcel)
          app.post('/thermalmodel/compute', this.computeController.computeModel)
      }
  }
  ```

- L'utilisation de « **PostMan »** pour simuler l'envoi client et vérifier le retour du solveur (ici le script).

- Les mécanismes asynchrones de NodeJS permettant d'éviter le caractère bloquant du mécanisme synchrone et par l'utilisation des fonctions « **CallBack** » qui donnent la possibilité de signaler le retour attendu de la fonction fournie en paramètre (voir annexe « Computecontroller »).

Une fois la route vers le solveur terminé, il était nécessaire de modifier les différentes fonctions synchrones par de l'asynchrone afin d'éviter tout processus bloquant.
J'ai donc changé la fonction « readfilesync » par « readfile » et « jsonstore » et ai ensuite géré les erreurs associées.

Readfilesync :

```
if (code === 0) {
  thermalModel.results = (JSON.parse(fs.readFileSync(path.resolve(runFolder,
'results.json')).toString()))
  res.send(thermalModel)
} else {
  res.status(500).send({
    message: 'solver failed',
    details: message
  })
```

Readfile:

```
if (code === 0) {
  let pathResult = path.resolve(runFolder, 'results.json')
  fs.readFile(pathResult, function read(err: any, data: any) {
    if (err) {
      throw err
    }
    jsonResult = data.toString()
    thermalModel.results = JSON.parse(jsonResult)
    res.send(thermalModel)
  })
} else {
  res.status(500).send({
    message: 'solver failed',
    details: message
  })
```

2.3 Développer des composants d'accès aux données

Les composants d'accès aux données sont réalisés à partir d'Express, en utilisant la bibliothèque npm XLSX pour la lecture et la sauvegarde du fichier Excel. L'échange des données entre la partie API et CLIENT est effectué à partir des méthodes *GET* et *POST* en utilisant les méthodes « REQUEST » et « RESPONSE» d'Express, les données sont transmise par le biais des variables « req.body » et « res.send » voir ci-dessous.

```
readResults(req: Request, res: Response, next: NextFunction) {
    const thermalModel = req.body
    let runFolder = process.env.RUN_FOLDER as string
    let pathResult = path.resolve(runFolder, 'results.json')
    fs.readFile(pathResult, function read(error: any, data: any) {
      if (error) {
        next(ErrorMessage.create("Error while reading results", {error: error}))
      }
      //recover result to thermalModel
      thermalModel.results = JSON.parse(data.toString())
      res.send(thermalModel)
    })
  }
}
```

2.3.1 Méthode de validation du fichier Excel

Une fois les deux modèles de conversion terminées, j'ai dû vérifier la validité du fichier XLSX., en passant en paramètre de la méthode un fichier XLSX retournant une erreur si le fichier ne contient pas toutes les feuilles attendues ou qu'une des feuilles ne contient pas toutes les colonnes attendues (voir Annexe validation du fichier XLSX).

2.4 Développer la partie front-end d'une interface utilisateur web NOS Partie Client

Une fois ces tâches achevées, j'ai pu commencer la partie Client « FRONT END ». Le Product Owner m'a confié le branchement de « *SendToSolver* » avec la route précédemment implémentée. Pour cela, j'ai dû me familiariser avec l'architecture Angular.

ANGULAR est un Framework côté client open source basé sur TypeScript. Il correspond à une réécriture complète d'AngularJS.

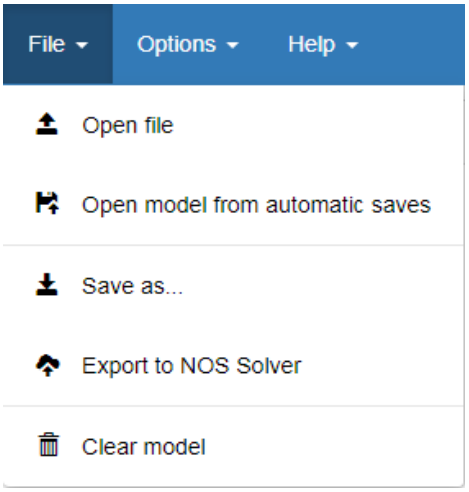
2.4.1 L'Exécution du calcul

Pour l'exécution du calcul j'ai commencé par modifier le model service afin d'appeler la route « /model/compute », puis générer la requête http avec la création d'une méthode « POST » à l'adresse localhost : 3000.

```
"/api/*": {
    "target": "http://localhost:3000",
    "secure": false,
    "logLevel": "debug",
    "changeOrigin": true,
    "pathRewrite": {
        "^/api": ""
    }
}
```

Une mise à jour du model à été nécessaire pour qu'il soit à même de recevoir les résultats. Et j'ai ensuite modifié le menu « navbar » pour appeler cette nouvelle route dans le service.

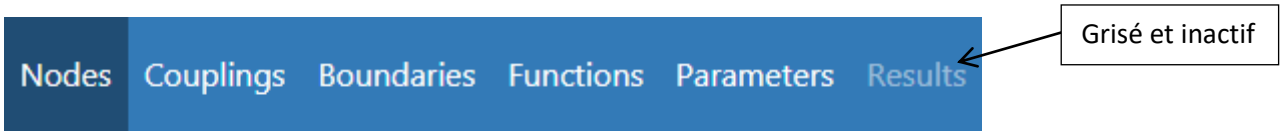
```
<li role="menuitem"><a class="dropdown-item" (click)="compute()">
    <span class="fa fa-cloud-download" aria-hidden="true"></span>
    Export to NOS Solver</a></li>
```



2.4.2 Vue résultats

Pour afficher les résultats il m'était nécessaire d'ajouter un nouvel onglet au menu, qui doit être désactivé si le modèle ne contient pas de résultat.

```
<li routerLinkActive="active" class="nav-item">
    <a class="outlets-result nav-link"
       [class.disabled]="!resultsAvailable"
       [class.opacite]="!resultsAvailable"
       [routerLink]="[{ outlets: { list: ['result'], detail: ['result']} }]">Results</a>
</li>
```



J'ai ensuite crée une route "results" dans le module de routing, j'ai créé un composant "resultList" contenant un tableau destiné à accueillir les résultats des calculs. Et enfin la création d'un abonnement à la réception de la réponse du à la route "compute", et naviguer vers la vue "results".

```
{
    path: "result",
    component: EmptyComponent,
    outlet: "list"
},
{
    path: "result",
    component: DisplayResultsComponent,
    outlet: "detail"
},
```

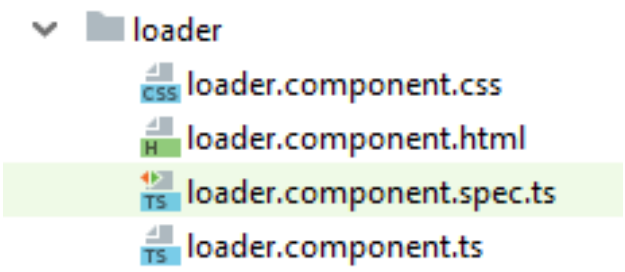
Results

Node ID	Node name	Temperature (°C)
1	Node 1	5
2	Node 2	10
3	Node 3	15
4	Node 4	20

2.4.3 Loader

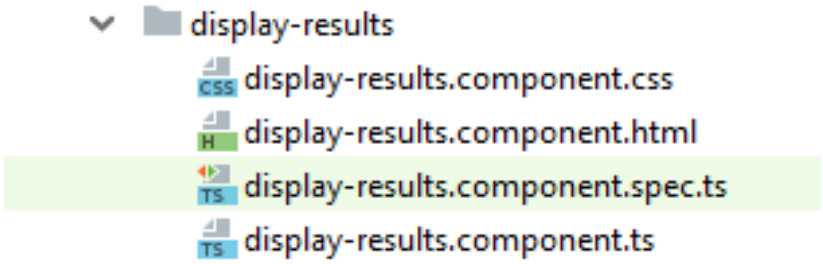
Lors de l'envoi du model au solveur un temps de réponse et de calcul est nécessaire, il est était donc important de le signaler au client par un message. Pour cela j'ai :

- Créé un composant Loader contenant une fenêtre modale et un GIF.




```
<div bsModal #template="bs-modal" class="modal fade in show " tabindex="-1" role="dialog"
     aria-labelledby="loaderModal" aria-hidden="true">
    <div class="loader modal-dialog modal-lg">
        <div class="modal-content container-fluid">
            <div class="modal-header modal-header-normal">
                <h4 class="modal-title pull-left">Loading</h4>
            </div>
            <div class="modal-body text-center">
                <img src="../../../assets/img/loading-gears-animation-12-2.gif"
alt=""/><br/>
            </div>
        </div>
    </div>
</div>
```

Le composant est abonné à la variable « *loading* » du service permettant d'ouvrir et fermer la « *Popup* » pendant l'attente de l'envoi du model au solveur dans mon cas.



- Crée un service pour gérer l'ouverture et la fermeture de la fenêtre modale.

J'ai créé un composant « *interceptor* » à l'aide du Product Owner, permettant d'intercepter toutes les requêtes faites par Angular au départ et au retour.

```
import ...
@Injectable()
export class LoaderInterceptor implements HttpInterceptor {
    constructor(public loaderService: LoaderService) { }
    intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
        this.loaderService.show();
        return next.handle(req).pipe(
            finalize( callback: () => this.loaderService.hide())
        );
    }
}
```

Il est très courant en Angular d'abonner un composant à un service pour effectuer certaine action après des événements. C'est normalement dans le service qu'on place toute l'intelligence métier.

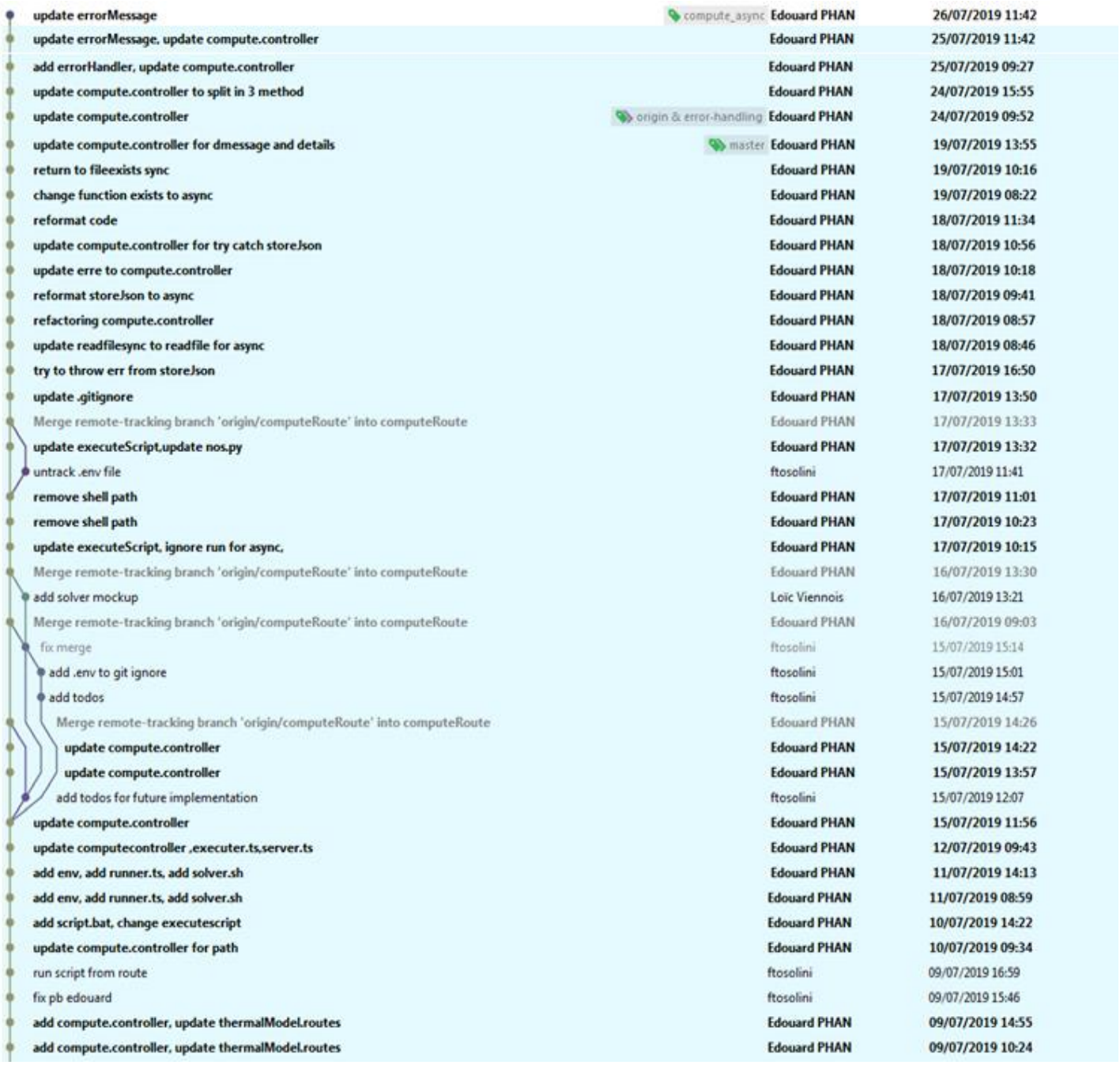
2.4.4 Migration vers Bootstrap 4

J'ai ensuite repris et modifié les fichiers .HTML afin d'adapter la nouvelle version de Bootstrap (version 4). Pour cela j'ai recherché les classes qui ne sont plus implémentés dans la dernière version de Bootstrap. Et j'ai modifié les fichiers .html pour adapter l'affichage et j'ai également repris les Glyphicons[5] pour les remplacer par des Font Awesome[6].

2.5 Suivi du projet

Le suivi a été réalisé à l'aide de Trello et GitLab. Vous retrouverez un aperçu du git log ci-dessous pour la partie Front-end et Back-end.

2.5.1 BACK-END



update errorMessage	compute_async	Edouard PHAN	26/07/2019 11:42
update errorMessage, update compute.controller		Edouard PHAN	25/07/2019 11:42
add errorHandler, update compute.controller		Edouard PHAN	25/07/2019 09:27
update compute.controller to split in 3 method		Edouard PHAN	24/07/2019 15:55
update compute.controller	origin & error-handling	Edouard PHAN	24/07/2019 09:52
update compute.controller for dmessage and details	master	Edouard PHAN	19/07/2019 13:55
return to fileexists sync		Edouard PHAN	19/07/2019 10:16
change function exists to async		Edouard PHAN	19/07/2019 08:22
reformat code		Edouard PHAN	18/07/2019 11:34
update compute.controller for try catch storeJson		Edouard PHAN	18/07/2019 10:56
update erre to compute.controller		Edouard PHAN	18/07/2019 10:18
reformat storeJson to async		Edouard PHAN	18/07/2019 09:41
refactoring compute.controller		Edouard PHAN	18/07/2019 08:57
update readfilesync to readfile for async		Edouard PHAN	18/07/2019 08:46
try to throw err from storeJson		Edouard PHAN	17/07/2019 16:50
update .gitignore		Edouard PHAN	17/07/2019 13:50
Merge remote-tracking branch 'origin/computeRoute' into computeRoute		Edouard PHAN	17/07/2019 13:33
update executeScript,update nos.py		Edouard PHAN	17/07/2019 13:32
untrack .env file		ftosolini	17/07/2019 11:41
remove shell path		Edouard PHAN	17/07/2019 11:01
remove shell path		Edouard PHAN	17/07/2019 10:23
update executeScript, ignore run for async,		Edouard PHAN	17/07/2019 10:15
Merge remote-tracking branch 'origin/computeRoute' into computeRoute		Edouard PHAN	16/07/2019 13:30
add solver mockup		Loïc Viennois	16/07/2019 13:21
Merge remote-tracking branch 'origin/computeRoute' into computeRoute		Edouard PHAN	16/07/2019 09:03
fix merge		ftosolini	15/07/2019 15:14
add .env to git ignore		ftosolini	15/07/2019 15:01
add todos		ftosolini	15/07/2019 14:57
Merge remote-tracking branch 'origin/computeRoute' into computeRoute		Edouard PHAN	15/07/2019 14:26
update compute.controller		Edouard PHAN	15/07/2019 14:22
update compute.controller		Edouard PHAN	15/07/2019 13:57
add todos for future implementation		ftosolini	15/07/2019 12:07
update compute.controller		Edouard PHAN	15/07/2019 11:56
update computecontroller ,executer.ts,server.ts		Edouard PHAN	12/07/2019 09:43
add env, add runner.ts, add solver.sh		Edouard PHAN	11/07/2019 14:13
add env, add runner.ts, add solver.sh		Edouard PHAN	11/07/2019 08:59
add script.bat, change executescript		Edouard PHAN	10/07/2019 14:22
update compute.controller for path		Edouard PHAN	10/07/2019 09:34
run script from route		ftosolini	09/07/2019 16:59
fix pb edouard		ftosolini	09/07/2019 15:46
add compute.controller, update thermalModel.routes		Edouard PHAN	09/07/2019 14:55
add compute.controller, update thermalModel.routes		Edouard PHAN	09/07/2019 10:24

2.5.2 FRONT-END

change glyphicon to fontawesome	origin & branch-display-results	Edouard PHAN	22/08/2019 13:24
update CSS		Edouard PHAN	20/08/2019 09:00
updatehtml, html, body { height: -webkit-calc(100% - 30px); }		Edouard PHAN	19/08/2019 13:40
add opacity to result navbar		Edouard PHAN	09/08/2019 13:27
change .html for adapt boostrap4		Edouard PHAN	08/08/2019 13:17
change navbar.component.html,coupling-detail,coupling-list,node-detail, node-list for adapt boostrap4		Edouard PHAN	07/08/2019 17:00
link disabled when no results		Edouard PHAN	07/08/2019 11:58
add compute response OK navigate to result tab		Edouard PHAN	07/08/2019 10:46
update display-result, clean app-routing, add getModel		Edouard PHAN	06/08/2019 14:36
change parameter-list to empty.component		Edouard PHAN	06/08/2019 13:45
update display-results		Edouard PHAN	06/08/2019 13:17
update app-routing, update thermalmodel, add result.model		Edouard PHAN	06/08/2019 11:47
update display-result component, update navbar		Edouard PHAN	06/08/2019 11:47
update LoaderModal	origin & branch-loader origin & dev	Edouard PHAN	06/08/2019 09:46
delete solverinfomodal		Edouard PHAN	06/08/2019 09:07
move <loader> change modal boostrap for loadermodal		Edouard PHAN	06/08/2019 08:17
move <loader>		Edouard PHAN	05/08/2019 15:33
clean loaderService, add LoaderInterceptor		Edouard PHAN	05/08/2019 15:00
try to add loaderService, add LoaderInterceptor		Edouard PHAN	05/08/2019 14:14
try to update loader.component		Edouard PHAN	05/08/2019 09:31
try to add loader.component		Edouard PHAN	02/08/2019 11:36
try to add loader.component		Edouard PHAN	02/08/2019 11:25
clean code	origin & branch-compute	Edouard PHAN	02/08/2019 08:50
clean code		Edouard PHAN	01/08/2019 14:02
clean code suprr sendtosolver		Edouard PHAN	01/08/2019 13:45
compute api call init		ftosolini	01/08/2019 11:22
add compute		Edouard PHAN	01/08/2019 10:01

3 Bilan et Perspectives

Le projet « Nodal Online Solver » NOS m'a permis d'avoir un aperçu des attentes et besoins clients en milieu professionnel. Cette application correspond à un besoin concret des ingénieurs thermiciens.

J'ai pris le temps de répondre aux attentes formulées par le Chef de projet et le Product Owner.

Tout d'abord, je pense que la PAE conduit en parallèle à la formation est une très bonne chose. Cela m'a donné l'opportunité de mettre en corrélation les compétences acquises durant la formation et les besoins concrets en milieu professionnel. Ce projet m'a permis d'améliorer mes compétences et d'en acquérir de nouvelles.

Je peux donc affirmer que ce projet m'a beaucoup apporté, j'ai eu l'opportunité de prendre de l'avance sur la formation en étudiant l'architecture N-tiers d'ANGULAR 9 et les tests unitaires avec Jest.

Ceci m'a conforté aussi bien dans la vision que dans la mise en pratique de mon futur métier de **C**oncepteur **D**éveloppeur d'**A**pplication.

J'aurais aimé avoir plus le temps pour améliorer l'Interface Homme Machine c'est-à-dire la partie CLIENT, en ajoutant divers fonctionnalités, ou bien d'optimiser l'ergonomie afin de faciliter le côté « *Users friendly* ». Hélas les contraintes inhérentes à la profession, ne m'ont pas permis de m'entretenir aussi souvent que nécessaire en direct avec les responsables de projet. Ce que je regrette, mais en contrepartie cela m'a permis d'affiner mes capacités de recherche.

Il me tarde de commencer la partie N-tiers pour une meilleure organisation en termes de code, et d'appliquer ou adapter les différentes compétences déjà acquises en milieu professionnel.

Je remercie toute l'équipe d'EPSILON pour son accueil, la mise en pratique des connaissances nécessaires pour devenir un **C**oncepteur **D**éveloppeur d'**A**pplication, et particulièrement le Product Owner Francis TOSOLINI et Loïc VIENNOIS pour l'enseignement technique et la vision.

4 Annexes

4.1 Dictionnary

```
export const boundaryAttributes = {
  'active': 'OFF',
  'node': 'Node ID',
  'value': 'Flux (W/m² or function)',}
export interface Boundary {
    active: boolean
    node: number
    value: number | string}
export const nodeAttributes = {
    'id': 'Node ID',
    'name': 'Node name',
    'cp': 'Capacitance (J/°C)',
    'multiplier': 'Multiplicator coefficient for capacitance',
    'tinit': 'Initial temperature (°C)',
    'x': 'X',
    'y': 'Y',
    'color': 'Color',}
export interface Node {
    id: number
    name: string
    cp: number
    multiplier: number
    tinit: number
    x: number
    y: number
    color: string}
export const couplingAttributes = {
    'active': 'OFF',
    'node1': 'Node ID 1',
    'node2': 'Node ID 2',
    'conductive': 'Conductive conductance  (W/°C or function)',
    'convective': 'Convective conductance  (W/°C or function)',
    'radiative': 'Radiative conductance (m² or function)',
    'advective': 'Advective conductance  (W/°C or function)',
    'multiplier': 'Multiplicator coefficient',}
export interface Coupling {
    active: boolean
    node1: number
    node2: number
    conductive: number | string
    convective: number | string
    radiative: number | string
    advective: number | string
    multiplier: number}
export const fluxBoundaryAttributes = {
    'active': 'OFF',
    'node': 'Node ID',
    'value': 'Flux (W/m² or function)',
    'multiplier': 'Multiplicator coefficient',}
export interface FluxBoundary  extends Boundary {
    multiplier: number}
export const powerBoundaryAttributes = {
    'node': 'Node ID',
    'active': 'OFF',
    'value': 'Power (W or function)',
    'multiplier': 'Multiplicator coefficient'}
export interface PowerBoundary extends Boundary {
    multiplier: number}
export const tempBoundaryAttributes = {
  'active': 'OFF',
  'node': 'Node ID',
  'value': 'Temperature (°C or function)'}
export interface TemperatureBoundary extends Boundary {}
```

4.2 Conversion

```
export function excel2Json (filePath: string) {
  const wb = readFile(filePath)
  const xlsxNodes = utils.sheet_to_json(wb.Sheets['Nodes'])
  if (excelConform(wb) && checkNodeIds(xlsxNodes)) {
    fileName = getFileName(filePath)
    saveThermalModel(wb, fileName)
  } else {
    console.log('Erreur le fichier XLSX n\'est pas conforme')
  }
}

function fromExcel (excelData: any, attributes: Attributes) {
  const newData = []
  attributes = invert(attributes)
  for (const item of excelData) {
    const newItem = mapKeys(item, (val, key) => {
      return attributes[key]
    })
    if (newItem.hasOwnProperty('active')) {
      newItem.active = newItem.active !== 'OFF'
    }
    newData.push(newItem)
  }
  return newData
}
```

4.3 Validation du fichier XLSX

```
export function convertNode (wb: WorkBook) {
  const xlsxNodes = utils.sheet_to_json(wb.Sheets['Nodes'])
  // @ts-ignore
  const nodes: Node[] = fromExcel(xlsxNodes, nodeAttributes)
  // tslint:disable-next-line:no-unused-expression
  const nNode = nodes.length
  for (let i = 0; i < nNode; i++) {
    if (nodes[i].id === undefined) {
      console.log('erreur manque un id ligne ' + (i + 1))}
    if (nodes[i].name === undefined) {
      console.log('erreur manque un nom ligne ' + (i + 1))}
    if (nodes[i].x === undefined) {
      console.log('erreur manque x ligne ' + (i + 1))}
    if (nodes[i].y === undefined && nodes[i].y !== null) {
      console.log('erreur manque y ligne ' + (i + 1))}
    if (nodes[i].tinit === undefined) {
      nodes[i].tinit = null}
    if (nodes[i].multiplier === undefined || nodes[i].multiplier === null) {
      nodes[i].multiplier = 1}}
  return nodes}

export function convertCoupling (wb: WorkBook) {
  const xlsxCouplings = utils.sheet_to_json(wb.Sheets['Conductances'])
  // @ts-ignore
  const couplings: Coupling[] = fromExcel(xlsxCouplings, couplingAttributes)
  for (let i = 0; i < couplings.length; i++) {
    if (couplings[i].active === undefined) {
      console.log('erreur manque un active true ligne ' + (i + 1))}
    if (couplings[i].node1 === undefined) {
      console.log('erreur manque un node1 ligne ' + (i + 1))}
    if (couplings[i].node2 === undefined) {
      console.log('erreur manque un node2 ligne ' + (i + 1))}
    if (couplings[i].multiplier === undefined || couplings[i].multiplier === null) {
      couplings[i].multiplier = 1}
    if (couplings[i].advective === undefined && couplings[i].radiative === undefined &&
      couplings[i].conductive === undefined && couplings[i].convective === undefined) {
      console.log('erreur manque un boundary ligne ' + (i + 1))}}
  return couplings }
```

```
export function convertFluxBoundary (wb: WorkBook) {
  const xlsxFluxBoundaries = utils.sheet_to_json(wb.Sheets['Flux'])
  const fluxBoundaries: FluxBoundary[] = fromExcel(xlsxFluxBoundaries, fluxBoundaryAttributes)
  let i
  for (i = 0; i < fluxBoundaries.length; i++) {
    if (fluxBoundaries[i].multiplier === undefined || fluxBoundaries[i].multiplier === null) {
      fluxBoundaries[i].multiplier = 1}}
  return fluxBoundaries}

export function convertPowerBoundary (wb: WorkBook) {
  const xlsxPowerBoundaries = utils.sheet_to_json(wb.Sheets['Power'])
  const powerBoundaries: PowerBoundary[] = fromExcel(xlsxPowerBoundaries, powerBoundaryAttributes)
  let i
  for (i = 0; i < powerBoundaries.length; i++) {
    if (powerBoundaries[i].multiplier === undefined || powerBoundaries[i].multiplier === null) {
      powerBoundaries[i].multiplier = 1}}
  return powerBoundaries}

export function convertTemperatureBoundary (wb: WorkBook) {
  const xlsxTemperatureBoundaries = utils.sheet_to_json(wb.Sheets['Temperature'])
  const temperatureBoundaries: TemperatureBoundary[] = fromExcel(xlsxTemperatureBoundaries, tempBoundaryAttributes)
  return temperatureBoundaries}

export function convertParameter (wb: WorkBook) {
  const xlsxParameter = utils.sheet_to_json(wb.Sheets['Simulation parameters'])
  const parameters: Parameter[] = fromExcel(xlsxParameter, parameterAttributes)
  return parameters}

export function saveThermalModel (wb: WorkBook, name: string) {
  const model = new ThermalModel(wb, name)
  const jsonPath = `${jsonOutputFolder}/${name}.json`
  storeJson(model, jsonPath)}

export function checkNodeIds (xlsxNodes: any) {
  const nodes: Node[] = fromExcel(xlsxNodes, nodeAttributes)
  const nodeIds = nodes.map((n) => n.id)
  return nodeIds.length === uniq(nodeIds).length}
```

4.4 Gestion des erreurs

```
import { Express } from 'express'

export default interface Middleware {
  applyMiddleware(app: Express): void
}
export function errorHandler(err: any, req: Request, res: Response, next: NextFunction) {
  let error: ErrorMessage
  if (!(err instanceof ErrorMessage)) {
    error = ErrorMessage.create("", {error: err})
  } else {
    error = err as ErrorMessage
  }
  console.log(error.error)
  res.status(error.status);
  res.send(error.toHttpResponse())
}
```

4.5 ComputeController

```
import {Request, Response} from 'express'
import {storeJson} from '../utils/file/file'
import {executeScript} from '../utils/executer/executer'
export class ComputeController {
    private test: string;
    constructor() {
        this.test = 'test'
    }
    computeModel(req: Request, res:Response) {
        const thermalModel = req.body
        console.log(thermalModel)
        // const jsonName = `${thermalModel.name}.json`
        const jsonName = `model.json`
        const jsonPath = `${process.env.RUN_FOLDER}/${jsonName}`
        storeJson(thermalModel, jsonPath)
        let childProcess = executeScript(
            process.env.SOLVER_PATH as string,
            process.env.RUN_FOLDER as string, jsonName,
            (code, message) => {
                if (code === 0) {
                    res.status(200).send({
                        message: 'solver ended successfully',
                        details: message
                    })
                } else {
                    res.status(500).send({
                        message: 'solver failed',
                        details: message})}})}}
```

4.6 ExecuteScript

```
export function executeScript(scriptPath: string,
                              folder: string,
                              jsonName: string,
                              callback: (code: number, message: string) => void) {
  const p = child_process.spawn(scriptPath, [jsonName], {
    cwd: folder,
    shell: process.env.SHELL_PATH,
  })
  let message = ''
  p.stdout.on('data', (data: Buffer) => {
    message += data.toString()
  })
  p.stderr.on('data', (err: Buffer) => {
    message += err.toString()
  })
  p.on('close', (code: number) => {
    callback(code, message)
  })
}
```

4.7 Abstract

Dans le cadre de la formation de **C**oncepteur **D**éveloppeur d'**A**pplication (**CDA**) au **C**entre de **R**ééducation **P**rofessionnel (**CRP**) de l'**I**nstitut **I**nformatique **S**ud **A**veyron (**2ISA**), J'ai effectué un projet suite à ma première période en entreprise chez EPSILON du groupe ALCEN dont vous trouverez le dossier projet ce rapport.
Ce projet a pour but de s'exercer dans le métier du numérique à travers une première expérience professionnel. Cela permet d'adapter les langages enseignés durant la formation, afin de s'adapter aux outils et langages les plus couramment utilisé en milieux professionnel.
L'entreprise EPSILON est spécialisée dans l'architecture des systèmes de hautes technologies Civils et Militaires, notamment dans le domaine Thermique.
Notre Chef de Projet Loïc VIENNOIS et le Product Owner Francis TOSOLONI nous ont confié le projet « NOS ».
Le projet NOS « NODALE ONLINE SOLVER » est une application Web destinée à modéliser des modèles thermique sous forme nodale. Il a pour but de répondre à la demande des ingénieurs thermicien. A l'origine l'application était en standby et nécessité donc une amélioration et des mises à jour.
Travailler sur cette application m'a permis d'appréhender les langages du type Type Script, Node.JS ainsi que la plateforme de développement Angular6 dont vous en trouverez un résumer dans ces écrits.

Si vous voulez en savoir plus, je vous invite à poursuivre plus loin….

As part of my training as a Software Developer in the Professional Rehabilitation Centre of 'Institut Informatique Sud Aveyron' (2ISA). I have carried out a project during my first practical training at EPSILON Company of the ALCEN Group which I describe in this report.

The goal of this project is an actual training among the IT profession through a professional experience. It gives me the opportunity to put into practice the languages that I have learned during my training, in order to adapt to the most commonly used tools and languages.

The EPSILON Company is specialized in high-tech Civil and Military systems architectures, especially in thermal engineering.

Our Project Manager Loïc VIENNOIS and the Product Owner Francis TOSOLONI entrusted us with the "NOS" Project.

The NOS project "NODAL ONLINE SOLVER" is a web application designed to model thermal models in nodal form. It aims to meet the demand of the thermal engineers. Originally the application was in standby and therefore needed improvements and updates.

Working on this application allowed me to understand the Type Script language , The NodeJS engine and the Front-end Framework Angular which you will find a summary in these writings.

If you want to know more, I invite you to read further…