République Algérienne Démocratique et Populaire
Ministère de l'enseignement Supérieur et de la Recherche Scientifique
Université de Batna 2 Mostefa Ben Boulaid

Faculté des Mathématiques et de d'Informatique

Département d'Informatique

Mémoire de Licence

Option : Systèmes Informatiques

Thème

Editeur de code

Présenté par :

Bouabdallah Azeddine

Mokeddem Ayoub

Dirigé par : Betta Mohammed

Promotion : 2017/2018

Dédicaces (Bouabdallah Azeddine)

Je dédie ce travail de fin d'études à :

Ma chère mère, mon père. Dont le mérite, les sacrifices et les qualité humaine qui m'ont permis de vivre ce jour. Je vous remercie pour tout le soutien et l'amour que vous me portez depuis mon enfance et j'espère que votre bénédiction m'accompagne toujours. Puisse Dieu, vous accorde santé, bonheur et longue vie et faire en sorte que jamais je ne vous déçoive.

Mon frère Younes et ma sœur Sara. Puisse Dieu leur prêterons longue vie et beaucoup santé et de bonheur.

Mon cousin Ahmed, qui m'a aidé plusieurs fois. Je vous souhaite le bon courage dans votre études.

Tous mes amis qui me soutiennent. Assaad, Oussama, Imad, Ayoub et Zakaria. En souvenir de notre sincère et profonde amitié et des moments agréables que nous avons passés ensemble.

Toute ma famille. Qui ont été toujours dans mon esprit et dans mon cœur,

Dédicaces (Ayoub Mokeddem)

Je dédie ce modeste travail a : Ma très chère et douce mère, Mon très cher père a qui m'adresse au ciel les vœux les plus ardents pour la Conservation de leur santé et de leur vie. Pour mes chers frères et sœurs. Pour mes très chers amis a toutes la promotion des 3eme années –System d'information - :2017-2018

Remerciements

En tout premier lieu, nous remercions le bon Dieu, tout puissant, de nous avoir donné la force pour survivre, ainsi que l'audace pour dépasser toutes les difficultés.

Nous remercions aussi Mr Ahmed Ghiloubi. Merci pour votre aide. Une chose est sure, nous n'aurons rien pu faire sans vous. Tout ce que vous avez fait signifie beaucoup pour nous.

Nous remercions aussi l'encadreur Mr Mohammed Betta. Nous vous remercions d'avoir partagé avec nous votre passion pour l'enseignement. Nous avons grandement apprécié votre soutien, votre implication et votre expérience tout au long de l'année.

Tables des matières (Sommaire)

Introduction générale

Depuis que le premier ordinateur a été inventé, des scientifiques, des ingénieurs et des informaticiens essayaient de rendre son utilisation de plus en plus facile et améliorer sa performance, l'ingénierie informatique s'améliore chaque jour, en conséquence les outils utilisés par les ingénieurs doivent être compatible avec le progrès des technologies.

De nos jours, le Web est devenu très important dans nos vies. Toutes nos activités sont maintenant liées au web (internet), ce qui a donné lieu à l'essor du développement web qui est devenu l'un des domaines les plus populaires de l'ingénierie informatique.

Aujourd'hui le développement web représente une grande partie de ce domaine d'étude, il augmente massivement par la montée de l'ingénierie informatique et internet. Le développement web nécessite la maîtrise des langages de programmation tel que html, css, javascript .. etc. le développeur web a besoin des outils qui faciliteront son travail car l'utilisation d'un éditeur de texte simple comme un bloc-notes peut être difficile, et prends beaucoup de temps.

Voici à quoi ressemble le code dans un éditeur de texte :

Code HTML

```
<!DOCTYPE html>
<head>
<title>Hello World</title>
<link rel="stylesheet" type="text/css" href="theme.css">
<body>
<h1>Hello World</h1>
<p>This is a very basic "Hello World" example made up in
HTML and CSS. See if you can change the size of the header
text above.</p>
</body>
</html>
```

Code CSS

```
body {
background-color: lightblue;
}
h1 {
color: white;
text-align: center;
}
p {
font-family: verdana;
font-size: 20px;
}
```

Figure 1. Le Code HTML et CSS dans un simple éditeur de texte

Pour cela, les développeurs cherchent un bon éditeur de code qui peut les aider à écrire un code facilement et à simplifier leur travail, en coloriant les balises et les mots clés sur leur code, donc pour cela on a choisi de travailler sur un éditeur de code pour rendre le développement web plus pratique et facile.

La figure suivante représente Comment le même code (représenté avant) ressemble à notre éditeur de code :

HTML Code

```html
<!DOCTYPE html>
  <head>
    <title>Hello World</title>
    <link rel="stylesheet" type="text/css" href="theme.css">
  <body>
    <h1>Hello World</h1>
    <p>This is a very basic "Hello World" example made up in
    HTML and CSS. See if you can change the size of the header
    text above.</p>
  </body>
</html>
```

CSS Code

```css
body {
    background-color: lightblue;
}

h1 {
    color: white;
    text-align: center;
}

p {
    font-family: verdana;
    font-size: 20px;
}
```

Figure 2. Le code HTML et CSS dans L'éditeur de code

Chapitre 1 : Définitions et analyse des besoins

I.1. Introduction

La définition et l'analyse des besoins est la première étape essentielle pour commencer à produire un projet, et pour faire une perspective de notre projet, et savoir où en va y aller lors de la conception et l'implémentation.

La définition des besoins d'un utilisateur aident les gestionnaires et développeurs du projet à bien comprendre à quoi sert cette application ou ce programme. Donc il est nécessaire de définir nos besoins avant le commencement du développement.

Notre objectif est de développer un éditeur de code, sans définir les besoins de celui-là, il est presque impossible de le réaliser complètement car on ne sait pas ce que nous devons faire pour finir la réalisation on voit aussi qu'il y a des éditeurs de code vraiment puissants, donc pour définir notre but et pour planifier notre démarche, on doit premièrement déterminer les objectifs de l'utilisation de l'éditeur et on va réaliser un simple éditeur de code qui va colorer la syntaxe de 3 langages et aussi qui permet de sauvegarder, ouvrir, créer des fichiers, et effectuer des opérations simples sur les textes. Ensuite on met tous les cas possibles de l'exécution de cet éditeur, comme les processus essentiels que le développeur va exécuter afin d'avoir une bonne expérience d'utilisation de l'éditeur personnalisé par les besoins donnés.

I.2. Définition des besoins

Pour mieux comprendre nos besoins on va faire comme un scénario d'exécution de l'éditeur pour bien déterminer et analyser les besoins.

I.2.1 Scénario

Azeddine est un développeur web qui veut développer un simple site web statique d'un hôtel. Premièrement Azeddine a créé une structure du dossier pour le site web. Dossier principal (Site Web) qui contient 2 sous dossiers (Images, Style) et 3 sous fichiers

(Index.html, Style.js) au même temps le dossier Style contient un seul fichier (Style.css). Dans l'ouverture du dossier dans l'éditeur, il affiche la liste des dossiers et sous dossier dans le côté gauche de l'interface (Site Web > 2 sous dossiers + 3 sous fichiers) et chaque sous dossier contient des sous fichiers. Azeddine veut débuter d'écrire le code html du site dans le fichier index.html. Il appuie sur Index.html dans la liste du projet à gauche et l'éditeur va ouvrir automatiquement le fichier pour l'éditer. Quand Azeddine veut sauvegarder les modifications il appuie seulement sur Cmd +S (Ctrl + S sous Windows), et l'éditeur va sauvegarder les modifications automatiquement. Quand Azeddine veut créer un nouveau fichier dans le projet, il peut juste appuyer sur Fichier => Créer un nouveau fichier et choisir l'emplacement du projet. Après quelque modification il se trompe et clique sur le bouton de fermeture sans sauvegarder le fichier, l'éditeur affiche une alerte pour lui informer. Des fois quand le fichier est très long l'éditeur bug des fois donc pour éviter ce problème l'éditeur doit contenir une fermeture de processus ? (CMD+Q sous mac et Alt+F4 sous Windows).

I.2.2 Besoins

Besoins fonctionnels

- Ouvrir des fichiers textes des plusieurs type (.txt, .html, .json, .css, …etc) ;
- Ouvrir des dossiers ;
- Créer des fichier texte (.txt, .html, .js, …etc) ;
- Supprimer des fichiers ;
- Ouvrir des multiples fichiers dans des plusieurs fenêtres dans l'éditeur ;
- Sauvegarder des fichiers dans l'emplacement actuel ou bien dans un emplacement spécifique ;
- Colorer la syntaxe du langage de balisage HTML et XML ;
- Colorer la syntaxe du langage de design CSS ;
- Colorer la syntaxe du langage de programmation JS ;
- Être capable de copier l'emplacement d'un fichier ;

Besoins non fonctionnels

- Application multiplateforme (Windows, Linux, Mac) ;
- Une Interface ergonomique et facile à utiliser.

I.3. Diagramme de cas d'utilisation

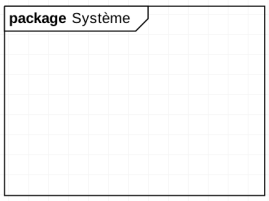
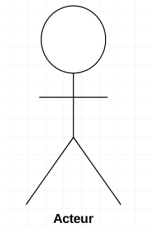
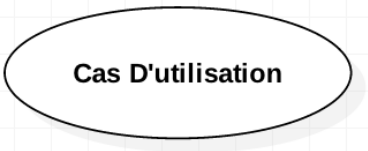
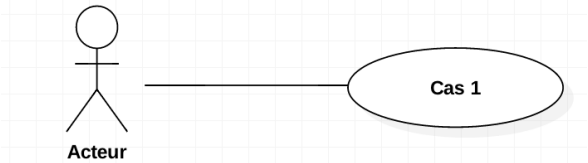
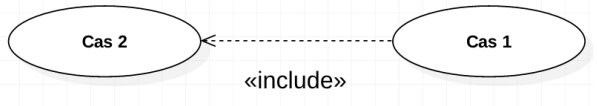
Le diagramme de cas d'utilisations est considéré comme la première étape du langage de modélisation UML pour modéliser les besoins des utilisateurs et identifier les fonctionnalités de base, les limites de système et représenter les interactions entre les utilisateurs et le système (Remy manu, cours d'uml). Ce diagramme nous aide à expliquer et à présenter nos idées avec une moyenne assez simple et claire, ceci aide le récepteur de comprendre notre système facilement.

Le diagramme de cas d'utilisation comporte 4 types de composants : un système, les acteurs, les cas d'utilisations et les relations entre les acteurs et le système. Le système représente votre application, programme, site web, etc. les acteurs ce sont tous ceux qui utilisent votre système. Ils peuvent être une personne, un autre système, une organisation, etc.

les cas d'utilisation permettent de représenter les fonctionnalités du système interne. Enfin il y a les relations. Elles peuvent être des associations, généralisations, associations et généralisation (Remy manu, cours d'uml).

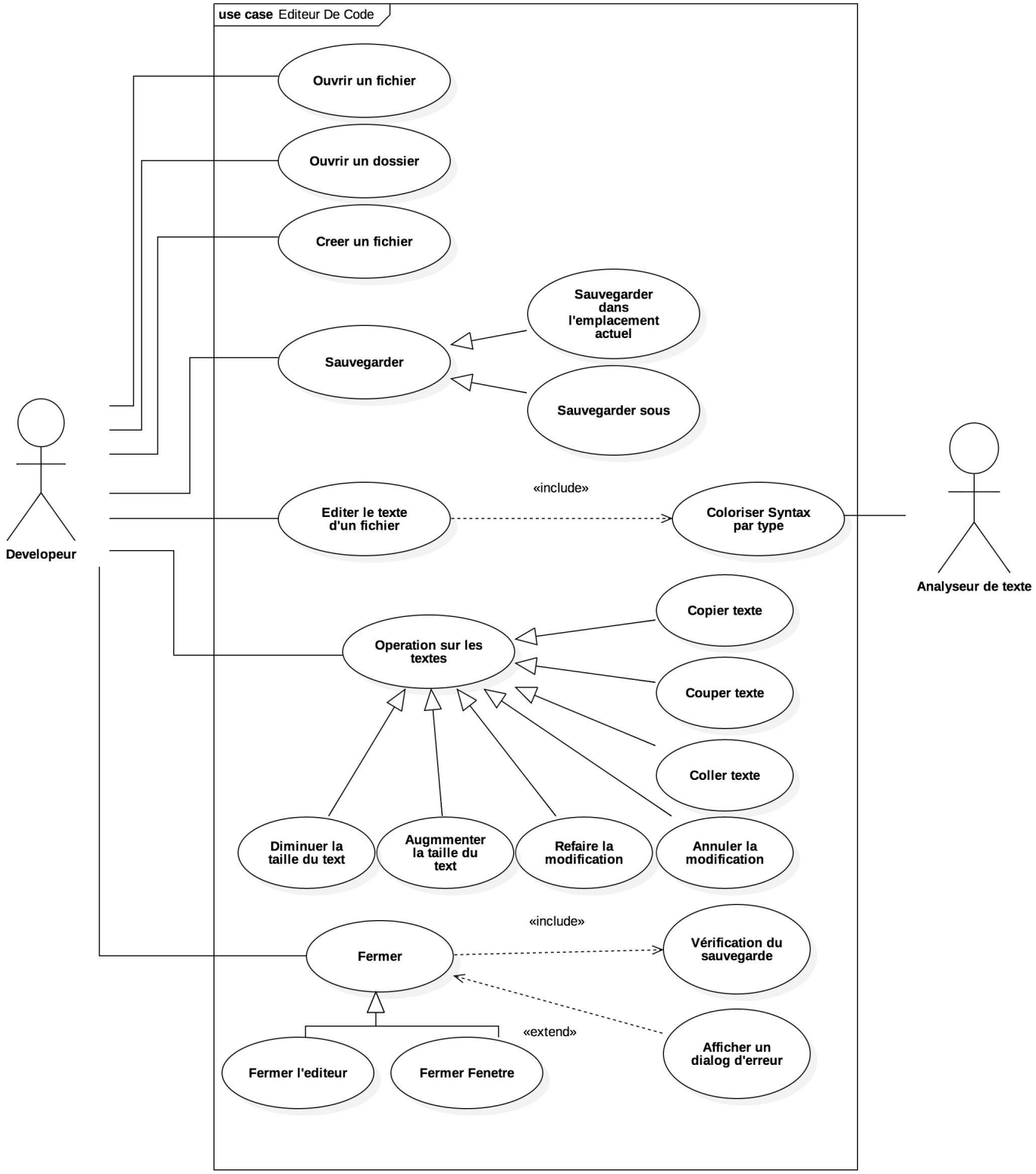
I.3.1. Notations

[1]

	**Système** : peut-être une application, un logiciel, un site web …etc.
	**Acteur** : tout réagit avec le système peut être une personne, autre système ...etc.
	**Cas d'utilisation** : toute fonctionnalité interne de système.
	**Association** : toute relation entre un acteur et une fonctionnalité dans le système interne.
	**Généralisation** : est une relation parentale entre les actions du système.
	**Extension** : est une relation entre une action et une autre. L'exécution de la première action exige l'exécution de la deuxième.
	**Inclusion** : est une relation entre deux actions ou plus, l'exécution de la première action peut suivi par la deuxième mais pas obligatoirement.

[1] Toutes les définitions dans le tableau sont depuis (Lucid Chart).

I.3.2. Diagramme :



I.4. Conclusion

Dans le premier chapitre de la définition et l'analyse des besoins, on a analysé notre projet et on spécifie nos besoins pour que ça soit clair et facile à réaliser.

La première étape est définir l'objectif et les fonctionnalités essentielle du projet. Notre éditeur de code doit être capable de gérer les fichiers et afficher leur contenu, colorer la syntaxe du texte. Ensuite on a utiliser le diagramme de cas d'utilisation pour bien expliquer les fonctionnalités du système et les acteurs principaux.

Chapitre 2 : Conception

II.1. Introduction

La conception est la partie la plus importante dans le processus de création d'un logiciel ou un programme, spécialement dans la création des logiciels et des modules complexes. Cette étape nous aide à faire un plan direct et un chemin clair pour notre trajet dans l'implémentation du projet. L'un des méthodes les plus utilisés dans nos jours est le langage de modélisation UML 2.0 (Smart Draw UML Documentation).

Ce langage est un moyen de décrire un logiciel en utilisant une collection de diagrammes (Smart Draw UML Documentation). Il nous aide de planifier notre projet sur des normes internationales, UML est tout un langage qui contient plus de 13 diagrammes utiliser pour la visualisation, dans notre conception du projet 'Editeur de code' on va utiliser le langage UML 2.0, mais on va utiliser seulement les 4 diagrammes suivants : Diagramme de cas d'utilisation, diagramme de classes, diagramme de séquence et diagramme d'activité.

II.2. Diagrammes de classes

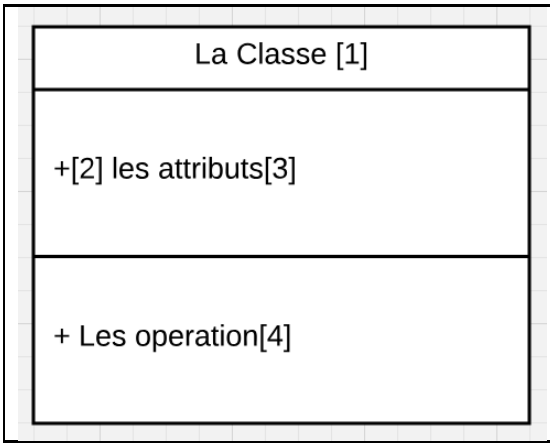
II.2.1. Définition

Le diagramme de classes est L'un des diagramme les plus populaires dans UML. il est utilisé pour documenter l'architecture de logicielle. Les diagrammes de classes sont un type de diagramme de structure puisqu'ils décrivent ce qui doit être présent dans le système en cours de modélisation.

Peu importe votre niveau de connaissance des diagrammes UML ou de classe. On utilise généralement le diagramme pour mieux comprendre l'aperçu général des schémas d'une application et illustrer des modèles de données pour les systèmes d'information en plus il permet de fournir une description indépendante de l'implémentation des types utilisés dans un système qui sont ensuite passés entre ses composants.

II.2.1. Notations

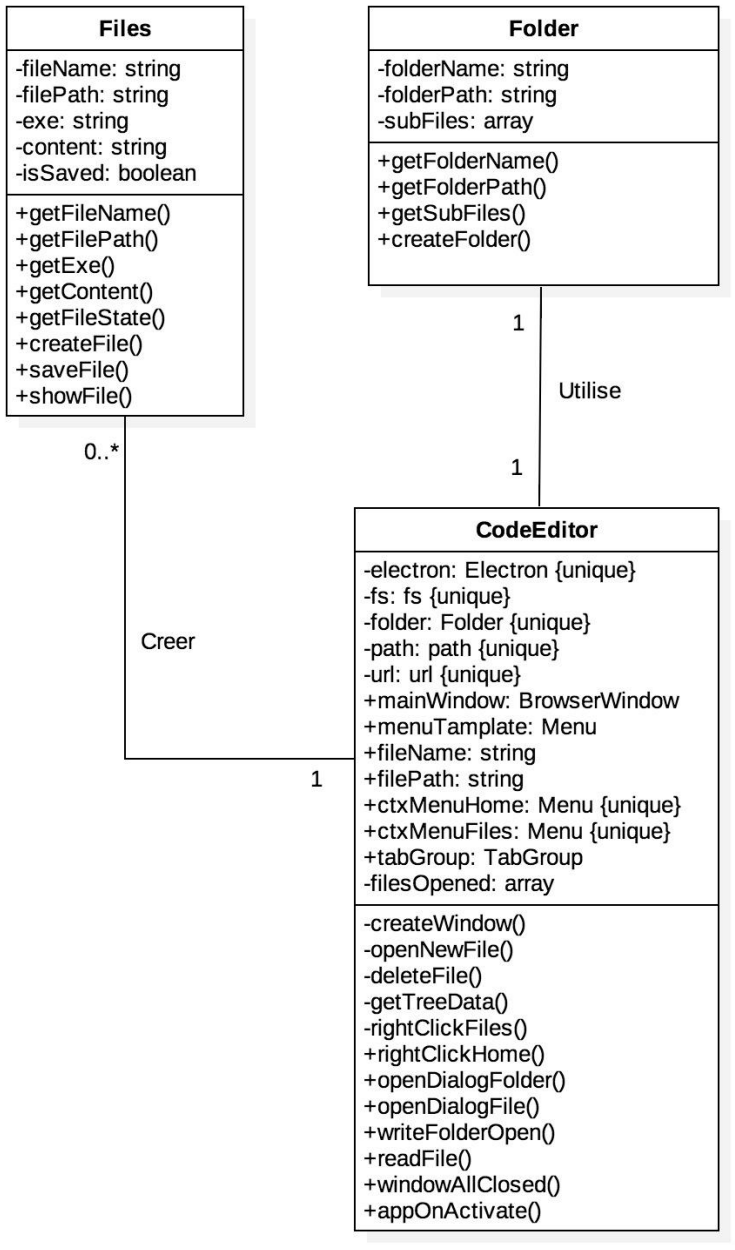
[2]

	**[1 Classe]** : Le nom de la classe utilisé
	[2 Visibilité] : Elle peut porter comme valeur (+) : publique, (-) : privé, (#) : protéger, (~) : paquet
	[3 Attribut] : Les attributs de la class. Sont écrits sous la forme : <Visibilité> <NomDeLattribut> : <Type>
	[4 Opération] : Les opération de la classe (Les méthodes et les fonctions). Sont écrit sous la forme : <Visibilité> <NomDeFonction>()

[2] Toutes les définitions dans le tableau sont depuis (Lucid Chart).

Verbe	**Association** : une relation entre les classes. Les deux classes sont conscientes l'une de l'autre et de leur relation avec l'autre [Ref : lucidchart.com]. Elle porte le verbe (type d'association) Exemple : utilise/ Fait appel à
0..*	**Multiplicité** : port comme valeur (1 : une seule fois), (1..* une seule fois ou plusieurs), (0..* : Zéro a plusieurs) (0..1 zéro a une seule fois) ou bien (n valeur fixe), (n..m intervalle)

II.2.2. Diagramme de classes



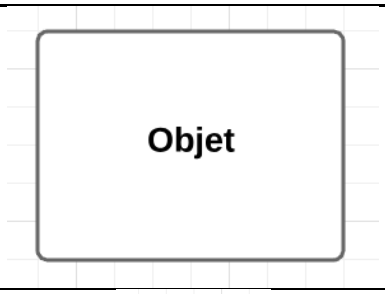
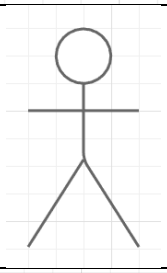
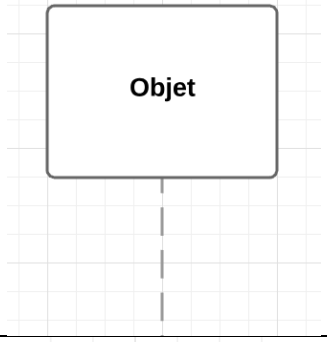
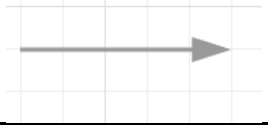
II.3. Diagrammes de séquences

II.3.1. Définition

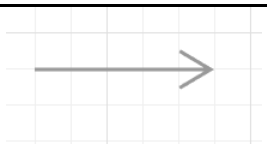
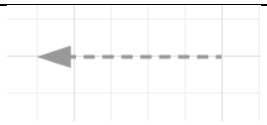
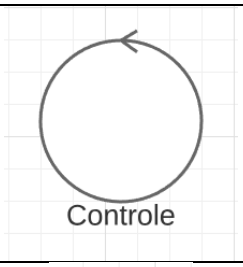
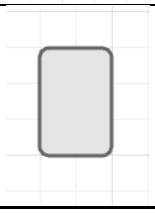
Un diagramme de séquence est un type de diagramme d'interaction puisqu'il montre comment les objets interagissent entre eux et l'ordre d'apparition. Ces diagrammes sont utilisés par les développeurs de logiciels et les professionnels pour comprendre les besoins d'un nouveau système ou documenter un processus existant. Les diagrammes de séquence sont parfois appelés diagrammes d'événements. (Lucid Chart UML tutorial) Les diagrammes de séquence peuvent être des références utiles pour les entreprises et autres organisations. On utilise ce diagramme pour représenter les détails du diagramme de cas d'utilisation, Il peut aussi modéliser la logique des fonctions, procédures et aussi les opérations sophistiquées, de plus en peut voir comment les objets et les composants interagissent les uns avec les autres pour compléter un processus (Smart Draw UML Documentation).

II.3.2. Notations

[3]

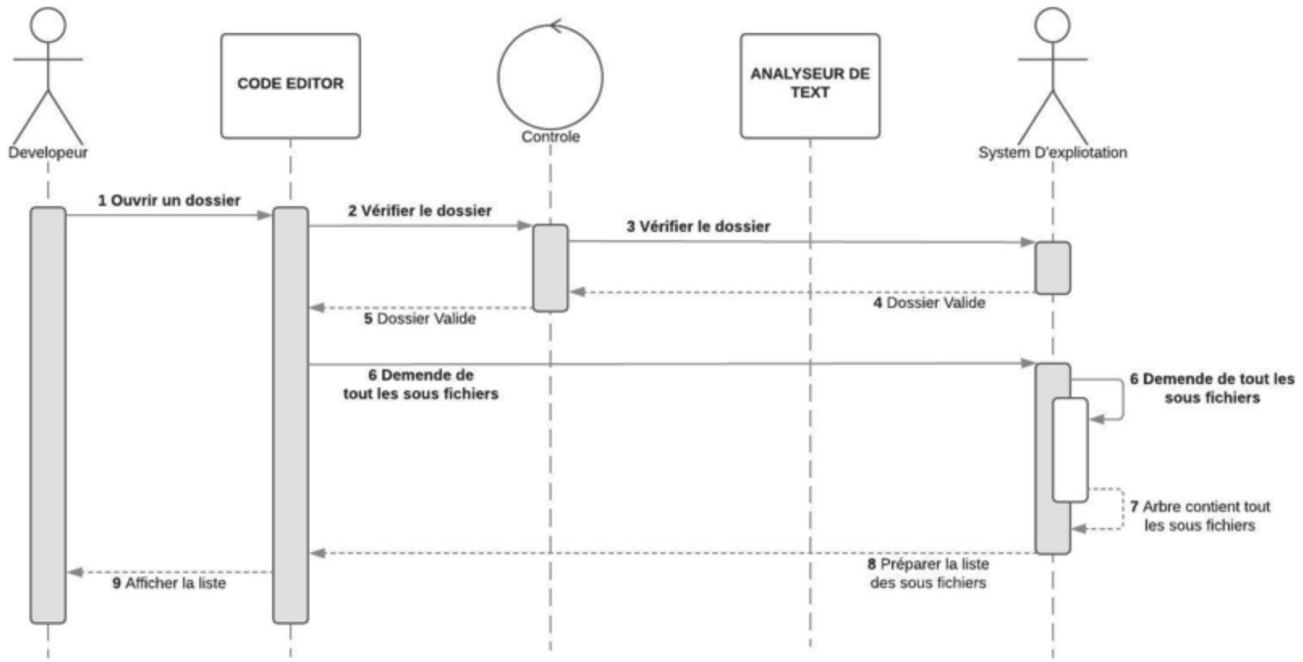
	**Objet** : représente une classe ou bien un objet dans le système, il montre comment un objet va se comporter dans le contexte de notre système
	**Acteur** : sont toutes les entités qui interagissent avec le système, ou sont externes au système
	**Ligne de vie** : Représente le passage du temps comme il s'étend vers le bas. Cette ligne verticale en pointillés montre les événements séquentiels qui se produisent sur un objet au cours du processus tracé. Les lignes de vie peuvent commencer par un rectangle étiqueté ou un symbole d'acteur.
	**Message synchrone** : Ce symbole est utilisé lorsqu'un expéditeur doit attendre une réponse à un message avant de continuer.

[3] Toutes les définitions dans le tableau sont depuis (Lucid Chart).

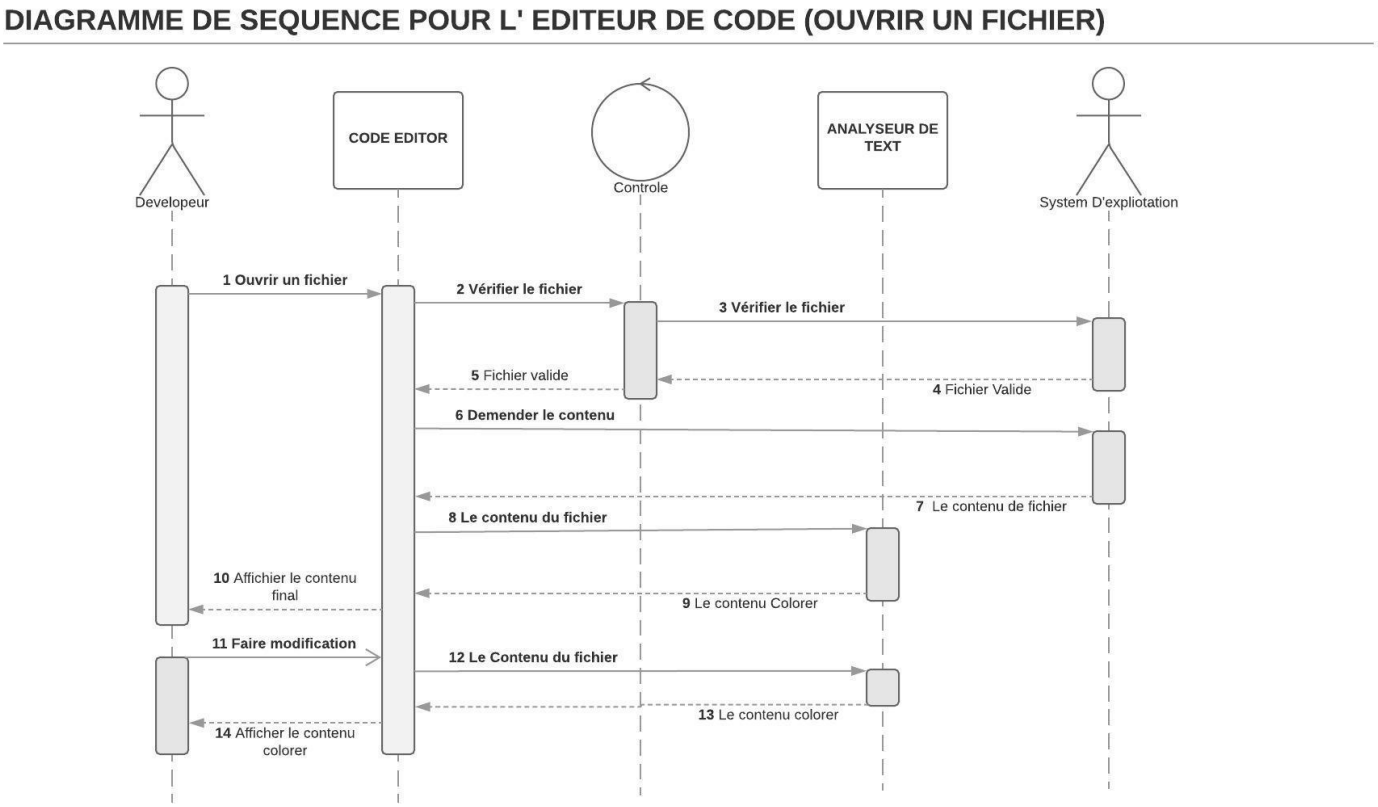
	**Message asynchrone** : ne nécessite pas de réponse avant que l'expéditeur ne continue.
	**Message de retour** : une réponse à un message reçu d'un autre objet ou acteur
	**Objet de contrôle** : simplement cet objet est responsable aux contrôles de notre système exemple : vérifier si le fichier existe si le dossier est valide, etc.
	**Boîte d'activation** : Représente le temps nécessaire à un objet pour terminer une tâche. Plus la tâche est longue, plus la boîte d'activation devient longue.

II.3.3. Diagramme de séquences 1

Diagramme de séquence pour l'editeur de code (Ouvrir un dossier)

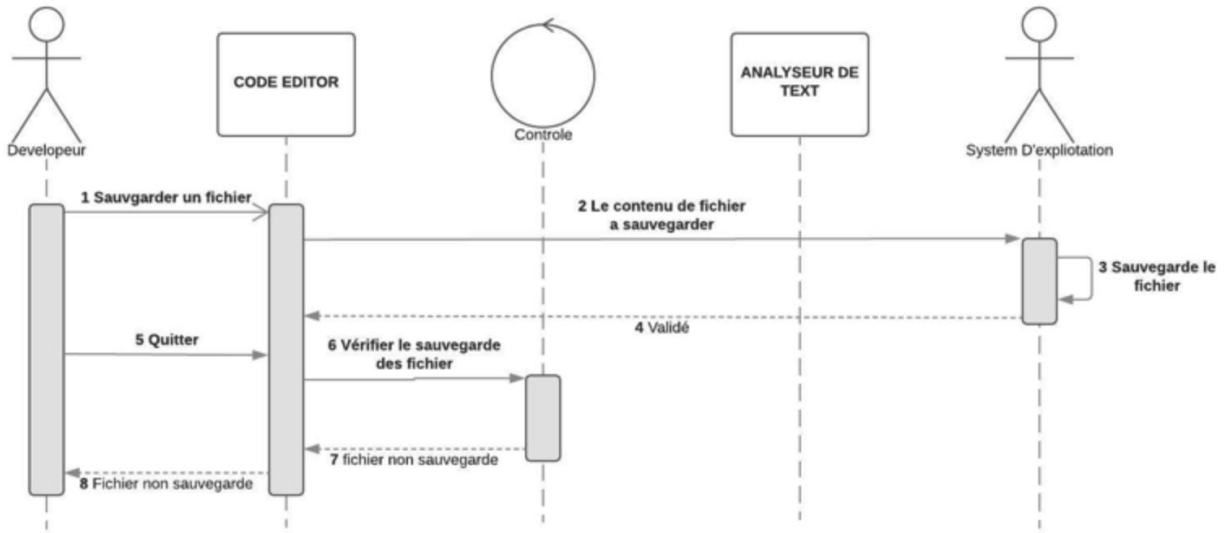


I.3.3. Diagramme 2



I.3.4. Diagramme 3

Diagramme de séquence pour l'editeur de code (Sauvegarder et Quitter)



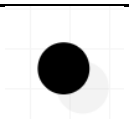
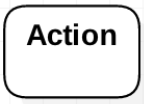
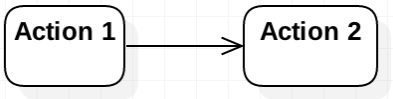
II.4. Diagrammes d'Activités

II.4.1. Définition

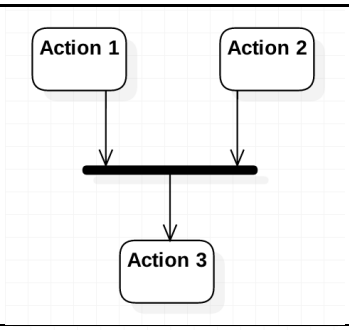
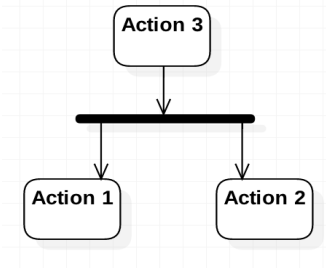
Les diagrammes d'activités, sont considérés comme un diagramme de comportement car ils décrivent ce qui doit se passer dans le système en cours de modélisation, et le présente graphiquement pour montrer le rôle de toute pièce ou composant dans le système. Parmi les avantages de diagramme d'activité : la démonstration de la logique d'un algorithme et le description des étapes effectuées dans un cas d'utilisation UML « Diagramme de cas d'utilisation », et l'avantage le plus important est de simplifier et améliorer n'importe quel processus en clarifiant les cas d'utilisation complexes. (Lucid Chart UML tutorial)

II.4.2. Notations

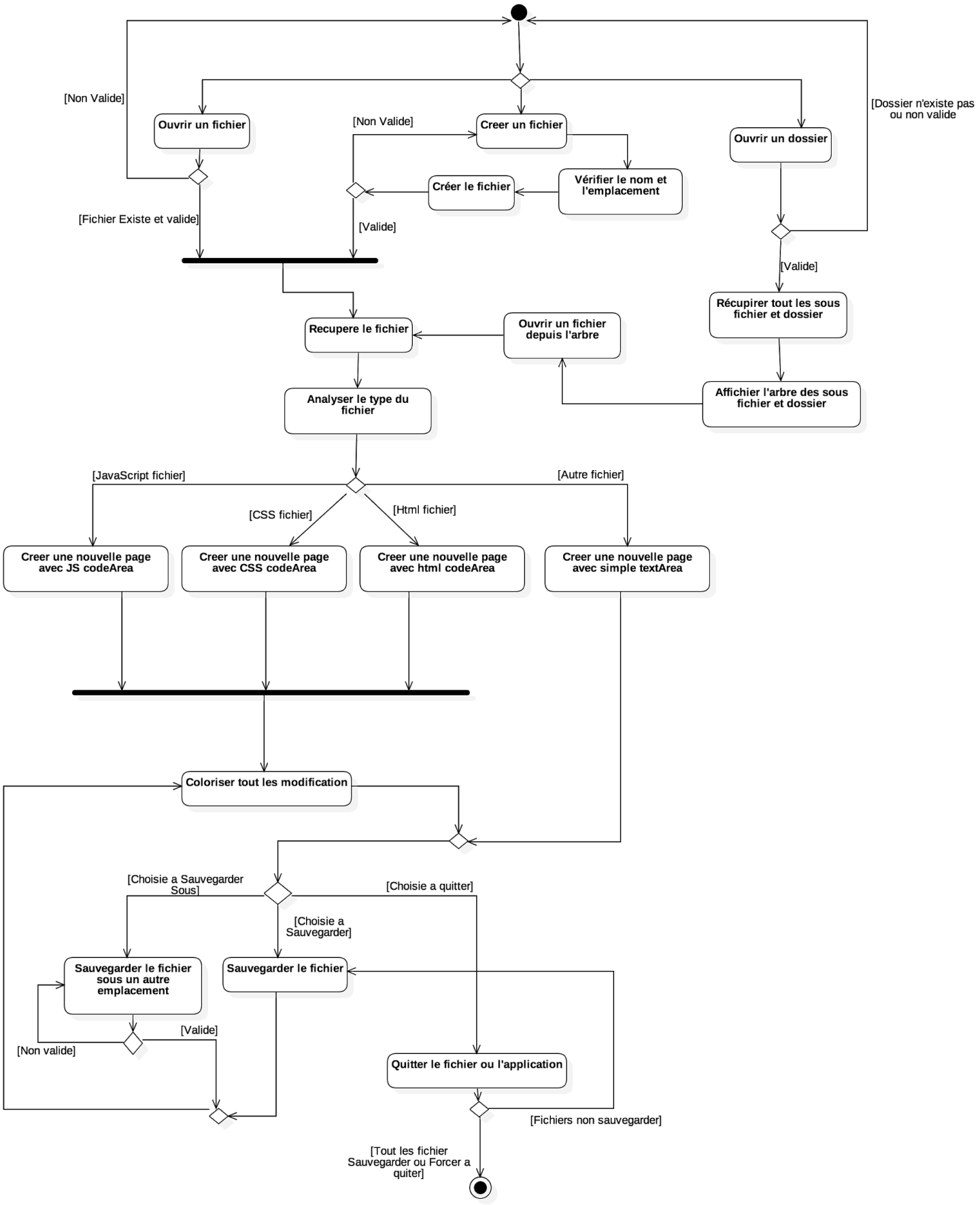
[4]

	**Début** : représente le début d'un processus
	**Fin** : Marque l'état final d'une activité et représente l'achèvement de tous les flux d'un processus.
	**Activité** : indique les activités qui comportent un processus modélisé, est le principal élément constitutif d'un diagramme d'activités.
	**Connecteur** : affiche le flux de contrôle de l'activité. Une flèche entrante commence une étape d'une activité, une fois l'étape terminée, le flux continue avec la flèche sortante.
	**Décision** : Représente une décision qui comporte au moins deux chemins débordant de texte de condition pour permettre aux utilisateurs d'afficher les options.
[Condition]	**Condition** : représente les conditions utiliser dans le symbole de décision

[4] Toutes les définitions de ce tableau sont depuis (Lucid Chart).

	**Barre de synchronisation** : Combine entre deux activités simultanées et les réintroduit dans un flux où une seule activité se produit à la fois.
	**Fork** : divise un flux d'activité unique en deux activités simultanées. Symbolisé avec plusieurs lignes fléchées d'une jointure.

II.4.2. Diagramme d'activités :



I.5. Conclusion

Dans le chapitre de la conception de notre éditeur de code on a utilisé le langage de modélisation UML 2.0 pour réaliser notre éditeur,t comme l'UML contient plus de 13 diagrammes au total, on a choisi d'appliquer 4 types de diagrammes seulement (Diagramme de cas d'utilisation, class, séquence, activité). On a représenté les différentes fonctionnalités de notre système sous la forme des diagrammes UML.

Chapitre 3 : Implémentation (Mise en œuvre)

III.1. Environnement de travail

L'éditeur de code a été développé pour être compatible avec tous les systèmes d'exploitation (Windows, Mac, Linux) et pour cela on a cherché un moyen pour que nous soyons capable de l'écrire un seul code pour tous les systèmes d'exploitations.

- Tout le processus de développement était sous le système d'exploitation macintosh.
- Le langage utilisé dans le développement de l'éditeur est JavaScript (ES6).

ES6 : ECMAScript est une spécification de langage de script standardisée par ECMAScript International. Il est utilisé par les applications pour activer les scripts côté client. Les langages comme JavaScript, JScript et ActionScript sont régis par cette spécification (ES6 tutorial documentation).

Dans notre projet on a utilisé des nouvelles technologies et frameworks pour les créations de softwares :

Electron.js : Notre principal outil de développement. Electron est un framework pour créer des applications natives avec les techniques web comme JavaScript, HTML et CSS avec tous les web frameworks comme Angular, React js, vue js, etc (Electron JS Documentation, 2013). Le travail principal d'Electron est basé sure le noyau de Chromium (Chromuim OS), et Node.Js

L'utilisation principale d'Electron js dans notre projet a était faite pour la création d'une application multi plateforme utilisant le langage JavaScript (ES6).

Node.js : est un moteur d'exécution JavaScript basé sur le moteur JavaScript V8 de Chrome. Le paquet de l'écosystème de Node.js « npm » est le plus grand écosystème de bibliothèques open source dans le monde. (Pour plus d'information sur le gestionnaire de paquets vous pouvez visiter le site officiel d'NPM : https://www.npmjs.com).

L'utilisation du Node.js est essentielle pour votre application si vous avez choisie d'utiliser Electron. Elle permette à votre application de communiquer avec le système et de faire des opérations au côté serveur (Back-End). Le travail de l'éditeur de code est basé sur les fichiers et les dossiers donc Node.js nous permettons d'obtenir les fichiers dans le système et des faires des opérations sur eux.

L'objectif principal de notre éditeur est de colorer la syntaxe des langages (Html, Css, JavaScript). Donc pour cela on a besoin d'un outil qui permet d'analyser le texte est de détecter les mots clé et de les colorer. Le meilleur choix pour cette mission est l'outil CodeMirror.

CodeMirror : est une zone de code (Code Area) implémenté en JavaScript pour les navigateurs. Il est spécialisé pour l'édition de code. Le travail principal de cette zone est d'analyser le texte entré à chaque fois et déterminer les mots clé et les colorer (CodeMirror).

On a adapté CodeMirror pour qu'il soit utilisé dans une application desktop au lieu d'un site web. Et aussi pour permettre à cet outils de communiquer avec les fichiers de système.

Bootstrap : est le framework le plus populaire pour la conception réactive des sites web, il permettra aussi des créer des beaux design (Bootstrap Framework).

On a utilisé bootstrap pour créer le design de la page detail.html, cette page comporte des informations générales sur l'éditeur.

JQuery : est une bibliothèque riche de Javascript qui permet de minimiser le code Javascript. Elle est utilisée dans le côté client 'Front-End' (JQUERY).

On a utilisé jquery pour capturer les évènements sur la page comme les évènements de cliques, les modifications, etc.

Split.js : est un framework qui permet de diviser la fenêtre aux plusieurs parties avec une fonctionnalité de glisser les fenêtres (Split.js framework).

L'utilisation principale de split.js dans notre projet est pour diviser la fenêtre en 2 parties, la partie projet et la partie espace de travail.

Sweet Alert 2 : est un module qui permette de créer des fenêtres de dialogue, et des messages d'information, d'erreur et d'information, etc. Il est utilisé pour electron.js (Sweet alert 2 github documentation)

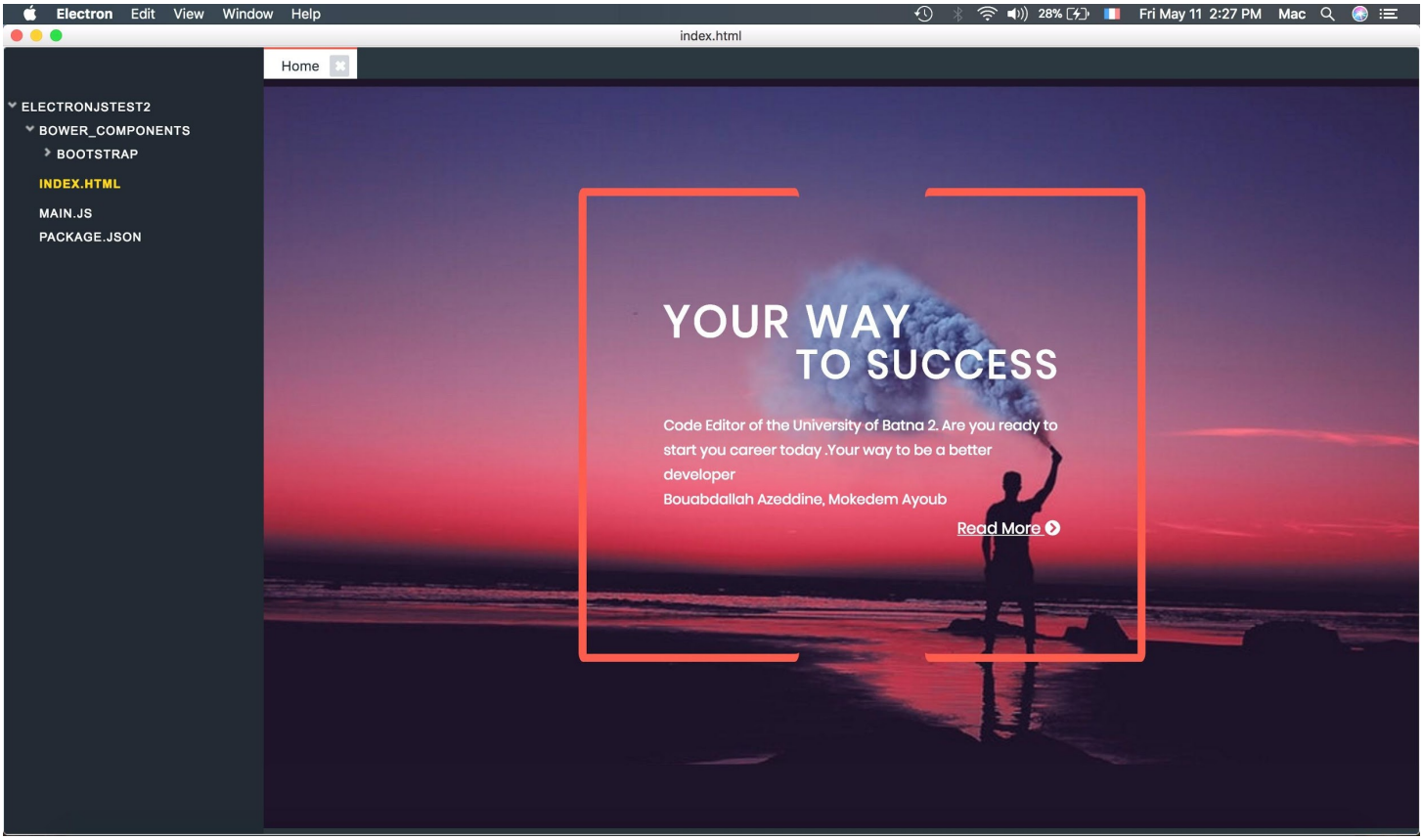
On a utilisé ce module pour afficher les messages de dialogues de la fermeture.

On a utilisé github [5] pour la procédure de développement et pour stocker le code. On a planifié de laisser le code Open Source. Vous pouvez visiter mon compte sure github : https://github.com/didoudesigner. Et vous pouvez aussi consulter notre code dans mon repository : https://github.com/didoudesigner/codeEditorJs .

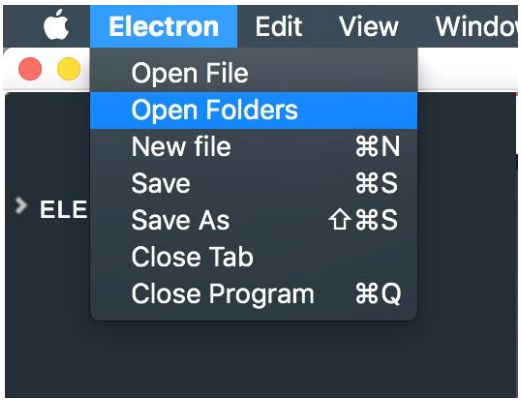
[5] Github est un outil gratuit pour héberger du code open source : https://github.com.

III.2. Description de l'application

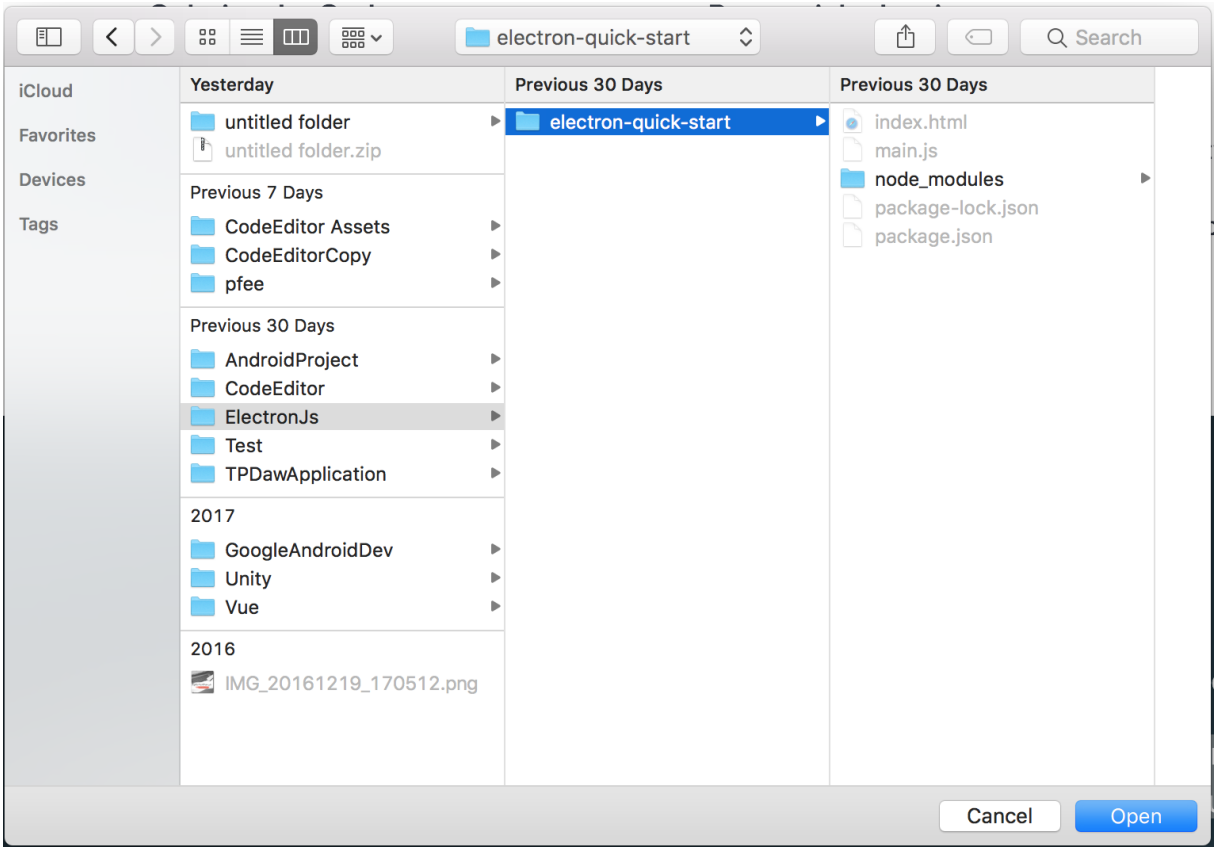
Dans le lancement de l'éditeur, la page d'accueil apparaît, vous pouvez lire à propos de l'éditeur dans la page des détails.



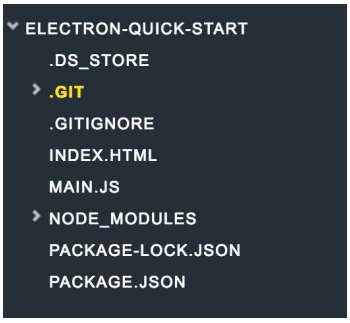
- Pour ouvrir votre premier projet. Cliquez sur le menu File (Electron pour Mac), ensuite Open Folder.



- Choisissez votre dossier. Ensuite la page va actualiser pour mettre votre dossier en place.

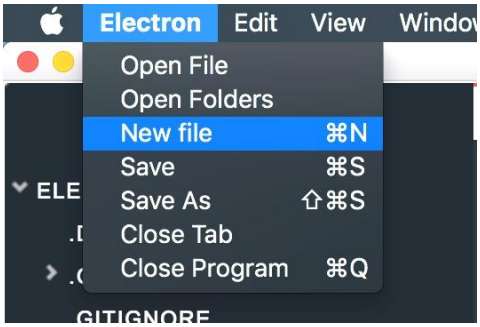


- Finalement vous pouvez ouvrir les fichiers du projet.

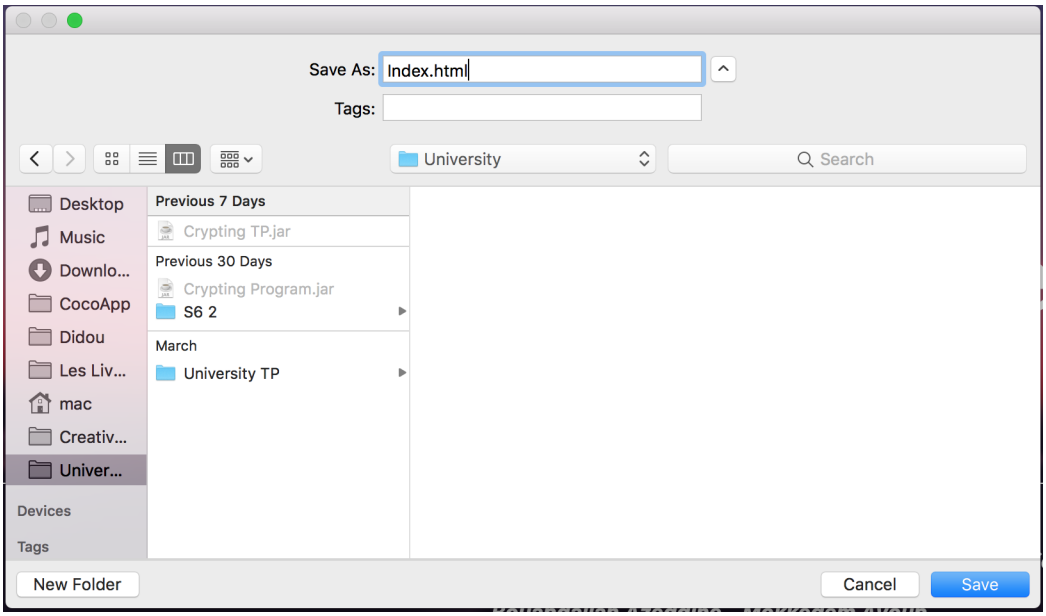


- L'éditeur contient un système des fenêtres pour faciliter l'édition de plusieurs fichiers au même temps.

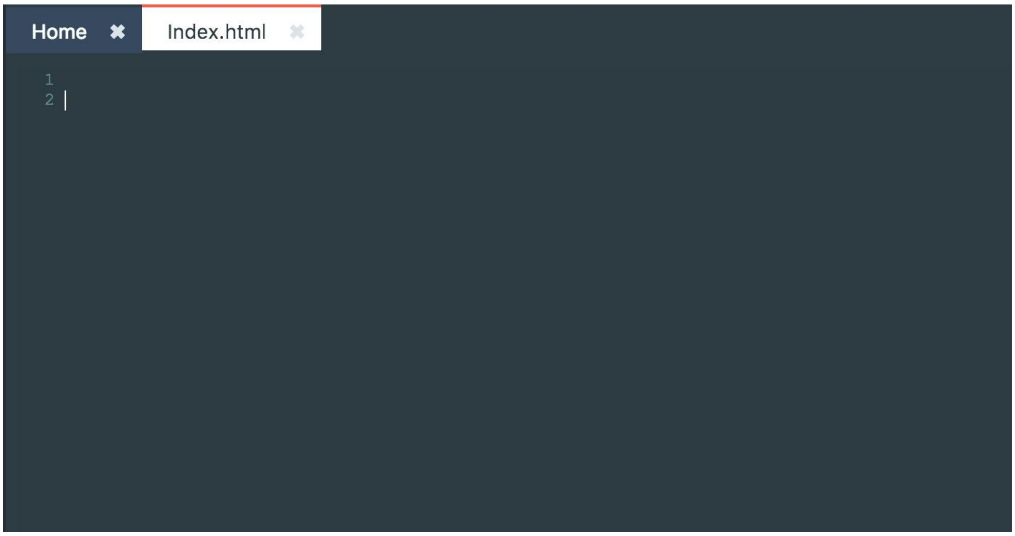
- Pour la création d'un fichier, il faut juste cliquez sure Electron (File) > New File.



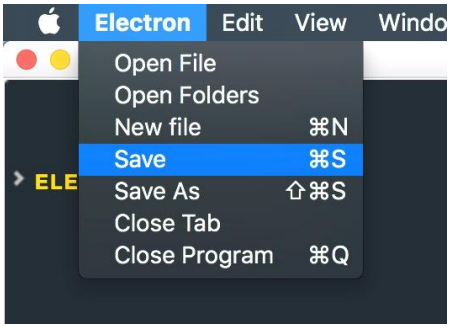
Ensuite choisissez l'emplacement du fichier.



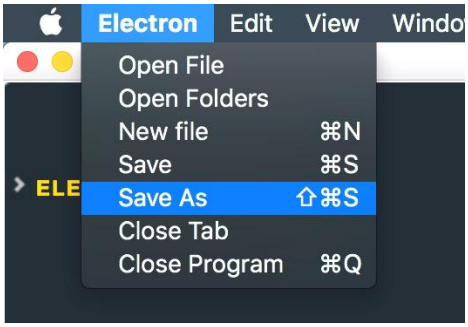
Enfin le nouveau fichier va être affiché.



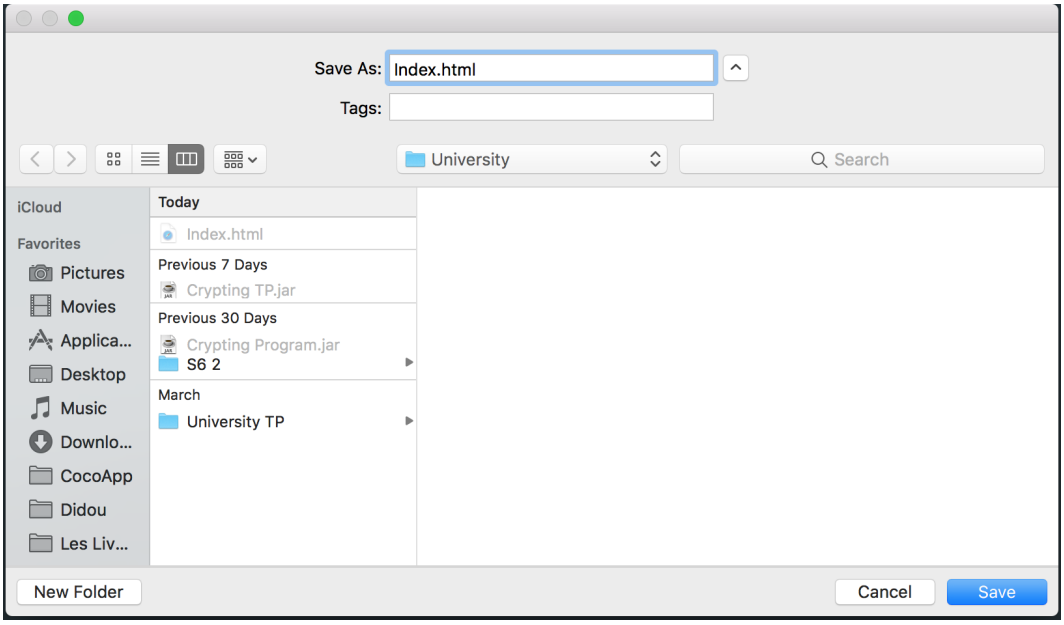
- Pour sauvegarder un fichier juste cliquez sur le menu Electron (File pour Windows) > Save



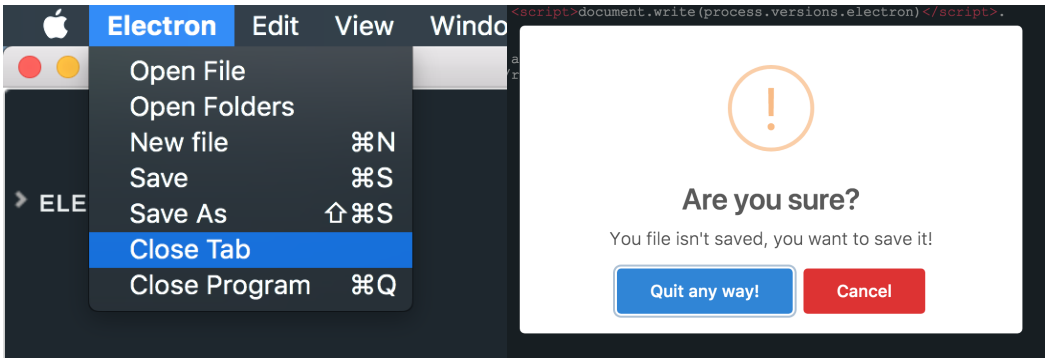
- Si vous voulez sauvegarder un fichier sous un emplacement spécifique cliquez sur le menu Electron (File pour Windows) > Save as.



Ensuite choisissez l'emplacement.

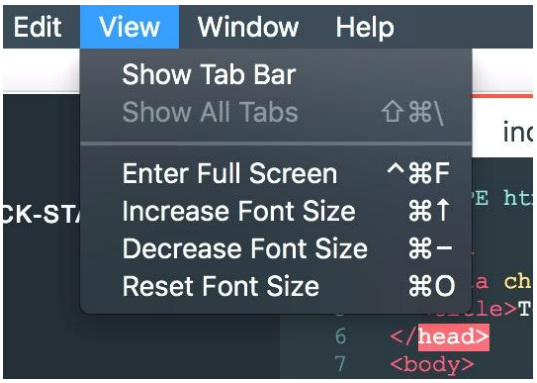


- La fermeture d'une fenêtre peut se faire en cliquant sur le bouton "X" ou bien dans le menu Electron (File pour Windows) > Close Tab. dans toutes les cas si votre fichier n'est pas enregistré un message vont apparaître pour la confirmation de la fermeture.

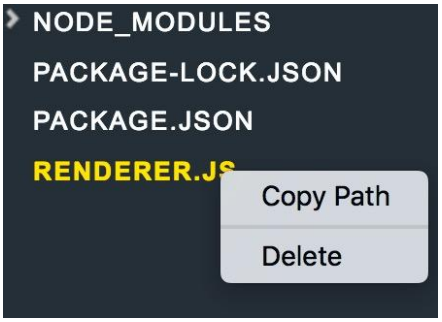


- Quand vous fermez l'éditeur complètement s'il y a un seul fichier n'est pas enregistré un message de confirmation apparaît. Par contre dans MAC si vous utilisez CMD+Q (Tuer le processus) même s'il y a un fichier n'est pas enregistrer l'éditeur va être fermé sans confirmation (ALT+F4 : dans le cas du Windows).

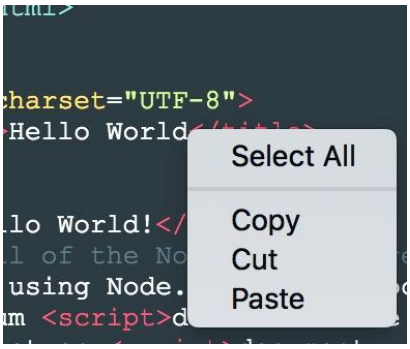
- Vous pouvez changer la taille de la police utiliser dans notre éditeur comme vous voulez. Juste cliquez sur View > Increase font size (pour incrémenter la taille), Decrease font size (pour Décrémenter la taille) ou bien Reset font size (pour remettre la taille par défaut).



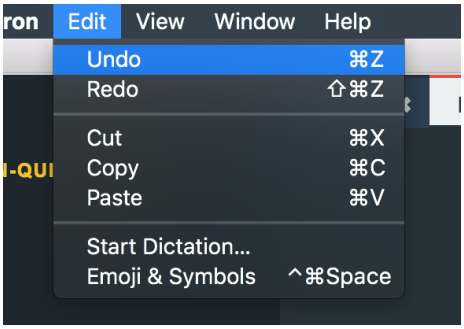
- Opération sur les Fichier dans L'arbre, Dans l'arbre des fichiers d'un certain dossier vous pouvez faires 2 opérations
  - Copy Path : qui va copier l'emplacement du fichier dans votre ordinateur
  - Delete : qui va supprimer le fichier complètement. cette opération va actualiser l'éditeur.



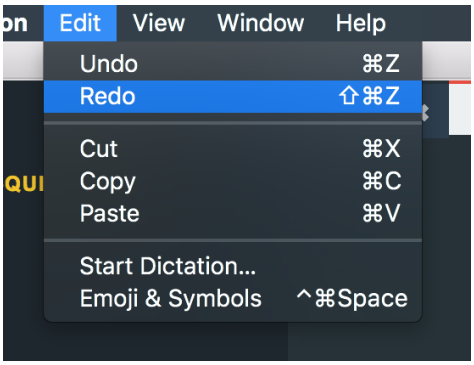
- Opération sur le texte dans l'éditeur, Dans l'éditeur vous pouvez faire plusieurs opérations sur le texte (le code). il y a 4 opérations :
  - Select All : permet de sélectionner tout le texte dans le fichier
  - Copy : Copier un certain texte sélectionné. S'il n'y a aucun texte sélectionné l'éditeur va prendre toute la ligne comme défaut
  - Cut : Couper un certain texte sélectionné. S'il n'y a aucun texte sélectionné l'éditeur va prendre toute la ligne comme défaut
  - Paste : Coller un texte déjà copier ou couper.



- Pour annuler des modifications sur le code (Annuler la frappe). Vous pouvez seulement taper sure Cmd + Z pour Mac et Ctrl + Z pour Windows.



- Pour refaire une frappe ou une modification sur le code vous pouvez seulement taper sure Cmd + Shift + Z pour Mac et Ctrl + Shift + Z pour Windows.



L'utilisation de notre éditeur est très simple et aussi nous avons ajouté des raccourcis clavier afin de donner une bonne expérience aux utilisateurs.

Conclusion générale

Le développement web s'est bien évolué, apprendre et maîtriser ce domaine est devenu une nécessité dans le marché de travail de nos jours. Donc on a choisi de réaliser un éditeur de code html, css et javascript, pour le but d'aider les développeurs web, et faciliter leur travail.

A partir de toutes les procédures de réalisation de notre projet 'Editeur de Code', nous avons appris plusieurs choses, à partir de la première étape de la définition des besoins jusqu'à la dernière étape de l'implémentation. On a appris comment préciser nos besoins dans un projet, comment utiliser le langage de modélisation UML pour planifier et expliquer le travail de toutes les parties de notre système. Et la meilleure partie de nos démarches est l'étape de l'implémentation. C'était la meilleure expérience dans notre formation. On a appris toutes les technologies utilisées dans notre projet tout seul.

Dans la réalisation de notre projet fin d'étude on a choisi de travailler en utilisant des méthodes modernes dans le domaine de développement de logiciels. Donc on a choisi d'utiliser le framework Electron.js qui utilise Node.js et Chromuim afin de construire une application desktop multi plateformes. On a adapté l'outil CodeMirror pour que soit utiliser dans notre application. On a la chance de réaliser tous nos besoins qui sont :

Ouvrir des fichiers textes des plusieurs type (.txt, .html, .json, .css, …etc).

- ✓ Ouvrir des dossiers.
- ✓ Créer des fichier texte (.txt, .html, .js, …etc).
- ✓ Supprimer des fichiers.
- ✓ Ouvrir des multiples fichiers dans des plusieurs fenêtres dans l'éditeur.
- ✓ Sauvegarder des fichiers dans l'emplacement actuel ou bien dans un emplacement spécifique.
- ✓ Colorer la syntaxe du langage de balisage HTML et XML.
- ✓ Colorer la syntaxe du langage de design CSS.
- ✓ Colorer la syntaxe du langage de programmation JS
- ✓ Être capable de copier l'emplacement d'un fichier.

Nous avons remarqué que les éditeurs modernes contiennent des fonctionnalités avancées, de la recherche jusqu'à la capacité d'exécuter des extensions externes comme des interpréteurs directs. Nous aimerions qu'on puisse dans le future d'améliorer notre éditeur pour qu'il soit capable d'intégrer des extensions et pour qu'il soit plus rapide.

Références bibliographiques

Bootstrap Framework. (s.d.). Récupéré sur Bootstrap Documentation: https://getbootstrap.com/docs/4.1/getting-started/introduction/

Chromuim OS. (s.d.). *Chommuim projects*. Récupéré sur https://www.chromium.org/

CodeMirror. (s.d.). *CodeMirror Acceuil*. Récupéré sur https://codemirror.net/

Electron JS Documentation. (2013). *Electron js documentation*. Récupéré sur Electron js: https://electronjs.org/docs

JQUERY. (s.d.). (D. Methvin, Éditeur) Récupéré sur JQUERY Api documentation: http://api.jquery.com/

Lucid Chart. (s.d.). *Lucid Chart UML tutorial*. Récupéré sur https://www.lucidchart.com/pages/what-is-UML-unified-modeling-language

Monte, L. (s.d.). *Sweet alert 2 github documentation*. (https://limonte.github.io/, Éditeur) Récupéré sur https://sweetalert2.github.io/

Smart Draw. (s.d.). *Smart Draw UML Documentation*. Récupéré sur Smart Draw: https://www.smartdraw.com/uml-diagram/

Split.js framework. (s.d.). *Split.js github documentation*. Récupéré sur https://nathancahill.github.io/Split.js/

Tutorial point. (s.d.). *ES6 tutorial documentation*. Récupéré sur Tutorial point: https://www.tutorialspoint.com/es6/index.htm