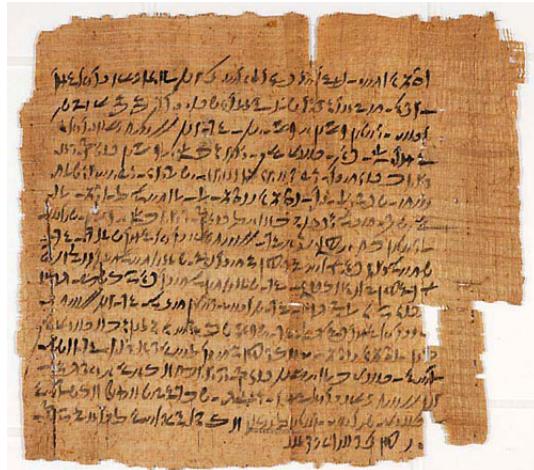


**RAPPORT DU PROJET DE PROGRAMMATION - MASTER 1
BIOINFORMATIQUE**

**Développement d'une Application Web de
Visualisation et de Manipulation de Fragments
de Documents Anciens**



GUEUX ELISABETH
MIALON SALOMÉ
PIET QUENTIN
POLIN AXEL

Client : M. ANTOINE PIRRONE

Résumé

La reconstitution de documents anciens, en particulier de papyrus, est un travail fastidieux pour les archéologues d'aujourd'hui. En effet, l'analyse d'un papyrus à des fins de recherche nécessite que celui-ci soit entier et en meilleur état possible afin d'en extraire le maximum d'informations. Des algorithmes de traitement et d'association d'images sont aujourd'hui en cours d'élaboration. Ceux-ci permettent d'effectuer une reconstitution de papyrus de façon automatique voire semi-automatique. Néanmoins, cette reconstitution nécessite un travail préalable de visualisation et de manipulation des différents fragments de la part des archéologues.

Une interface web pourrait alors offrir un service de visualisation, manipulation, et d'application de divers algorithmes pour répondre à leurs besoins. L'implémentation d'une telle application nécessite donc la programmation d'une interface fonctionnant sous un modèle client/serveur à partir duquel ils pourront avoir accès à diverses fonctionnalités. L'enjeux est donc de créer une interface web, pleinement fonctionnelle, offrant des services de manipulations pratiques d'images, une à une ou conjointement. Cette interface doit permettre de simplifier le travail de l'archéologue par l'application de divers d'algorithmes.

L'architecture de l'interface doit être réalisée de façon à ce que de nouvelles données (images ou xml associé) ou de futurs programmes de traitement d'image et d'assemblage puissent être implémentés au fil du temps.

Table des matières

Introduction	1
1 Analyse	3
1.1 Contexte	3
1.2 État de l'art	4
1.3 Analyses des besoins fonctionnels et non fonctionnels	5
1.3.1 Besoins Fonctionnels	5
1.3.2 Besoins non Fonctionnels	6
1.4 Choix de réalisation	7
2 Conception	9
2.1 Client	9
2.1.1 L'interface	9
2.1.2 Le Canvas	9
2.1.3 Le Formulaire	10
2.1.4 Création de répertoire	10
2.1.5 Sauvegarde d'un assemblage	11
2.2 Serveur	11
2.2.1 Gestion des données	11
2.2.2 Intégration et exécution des scripts	12
2.2.3 Gestion de la sécurité et des sessions personnelles	12
2.3 Architecture établie	12
3 Réalisation	14
3.1 Initialisation du Serveur	14
3.2 Accès à l'interface et authentification de l'utilisateur	14
3.2.1 Authentification	14
3.2.2 Ajout d'utilisateurs	15
3.3 Création et fonctionnement général du menu Canvas	15
3.3.1 Area	15
3.3.2 Area.images	16

3.4	Importation des données	16
3.4.1	Création et remplissage du panneau latéral	16
3.5	Affichage des fragments dans le canvas	17
3.5.1	Drag and Drop	17
3.5.2	Modification de l'affichage d'un fragment depuis la barre latérale	17
3.5.3	Mise à l'échelle réelle des images	18
3.6	Manipulation sur les fragments	19
3.6.1	Déplacement des images dans le canvas	19
3.6.2	Autres fonctionnalités : Barre d'outils	19
3.6.3	Application des algorithmes de Treshold et de Score d'assemblage	20
3.6.4	Affichage des méta-données	22
3.7	Création d'un répertoire de travail, enregistrement d'assemblages et manipulation des fichiers disponibles pour la session	22
3.7.1	Création d'un répertoire de travail	22
3.7.2	Suppression d'un répertoire de travail	23
3.7.3	Enregistrement d'un assemblage	23
3.7.4	Suppression d'images du serveur/répertoire	24
3.8	Mise en place d'éléments de sécurité	25
3.8.1	Utilisation exclusive du protocole HTTPS	25
3.8.2	Certificats de sécurité	25
3.8.3	Sécurisation des en-têtes HTTP	25
3.8.4	Protection contre les attaques CSRF	25
3.8.5	Protection des mots de passe	26
3.9	Manuel utilisateur	26
3.10	Perspectives	26
3.10.1	Modification des métadonnées	26
3.10.2	Options pratiques et d'ergonomie	26
3.10.3	Choix des images pour l'implémentation	27
Conclusion		28

Introduction

Dans le cadre de la Première Année du Master de BioInformatique, un Projet de Programmation doit être réalisé. Ce projet sera encadré par Monsieur Antoine Pirrone qui réalise sa thèse pour un projet de recherche scientifique, nommé GESHAEM (ou The Graeco-Egyptian State : Hellenistic Archives from Egyptian Mummies). Différents laboratoires sont associés à ce Projet dont le LaBRI (Laboratoire Bordelais de Recherche en Informatique). Antoine Pirrone travaille au sein de ce laboratoire sur le développement d'algorithmes d'associations d'images de fragments de papyrus.

Conjointement à ce projet, il a été proposé de développer une application web de visualisation et de manipulation de fragments anciens. En effet, le projet consiste à étudier un corpus papyrologique grec ou égyptien retrouvé sous forme de fragments dans des sarcophages inutilisables en l'état. Il est donc nécessaire de pouvoir les reconstituer afin de pouvoir extraire les informations portant sur la fiscalité et l'administration du IIIème siècle avant notre ère, de la région de Fayoum. Antoine Pirrone est donc en charge de développer des algorithmes qui permettront suivant la forme, la taille de l'écriture présente sur le Papyrus ou le découpage du morceau de papyrus (détection de contours) de créer des assemblages et d'avoir un papyrus reconstitué.

Pour le confort de visualisation et la simplicité d'utilisation, la réalisation d'une interface disponible pour les chercheurs est donc nécessaire. Cette interface devra permettre de visualiser un ou plusieurs fragments en même temps et permettre de les manipuler : zoomer ou dézoomer, pivoter sur la droite ou la gauche et les afficher en taille réelle. Il faudra aussi pouvoir visualiser les différentes versions du fragment : son recto, son verso ou sa version infrarouge ou en couleur. Un autre traitement d'image fourni par un algorithme, le threshold, devra être disponible. Il permettra de réaliser une opération de seuillage sur le fragment sélectionné. Un second algorithme sera fourni et renverra des scores de probabilité d'associations.

De plus, d'autres fonctionnalités seront accessibles. D'une part, l'utilisateur, pourra modifier les données associées à un fragment de papyrus. Il pourra aussi se créer une session pour enregistrer tous les assemblages ou manipulations/modifications qu'il aura entrepris. Il pourra alors les retrouver lors d'une prochaine connexion sur l'interface.

Concernant la programmation, l'interface sera développée en HTML5 (HyperText Markup Language 5), le langage de balisage standard pour la génération de page web. Il faudra rajouter du CSS3 (Cascading Style Sheets 3) pour développer les feuilles de styles de l'interface. Le framework W3.CSS sera utilisé en parallèle. Et enfin, le langage JavaScript sera utilisé de concert avec le framework AngularJS afin de développer le moteur de l'interface. Les scripts de threshold et d'assemblage seront implémentés en Python. De plus, un serveur en Node.js sera conçu.

Il s'agit ici de présenter le travail effectué afin de mener à bien la réalisation de cette interface Web.

Tout d'abord, une analyse complète du sujet sera effectuée pour rappeler le contexte du projet, une observation des interfaces graphiques et des outils de développement web existants sera faite. Afin de répondre au mieux aux attentes du client, une liste des besoins fonctionnels et non fonctionnels sera établie.

Par la suite une présentation détaillée du modèle de réalisation choisi sera fournie. Enfin, la réalisation de cette interface sera détaillée et illustrée par les résultats obtenus.

Chapitre 1

Analyse

1.1 Contexte

L’application à développer s’inscrit dans le projet du GESHAEM (ou The Graeco-Egyptian State : Hellenistic Archives from Egyptian Mummies). C’est un projet financé par le Conseil Européen de la recherche, sur cinq ans. Il est hébergé par l’institut Ausonius de l’Université de Bordeaux Montaigne.



FIGURE 1.1 – Logo du Projet GESHAEM

Les chercheurs veulent donc étudier des papyrus de la collection Jouguet conservés à la Sorbonne. Ils contiennent des informations administratives ou fiscales (factures) rédigées en égyptien démotique ou en grec démotique et ont une forte valeur historique pour les archéologues. Cette collection permettrait une meilleure connaissance de l’administration ptolémaïque Egyptienne. Le projet GESHAEM a pour but d’améliorer les connaissances concernant l’économie, la fiscalité et la gestion territoriale de la région du Fayoum, l’une des plus importantes régions agricoles d’Egypte.

Malheureusement ces papyrus ont été retrouvés en fragments dans les sarcophages en tant que cartonnages, servant notamment à la confection de masques funéraires pour les momies. Ils sont alors abîmés et il est difficile de les analyser dans l’état.

Pour reconstruire ces textes, grâce aux fragments, la mise au point d’un instrument numérique de traitement automatique de l’image pourrait grandement aider. Cela permettrait un assemblage des fragments plus facile.

Il y a alors le besoin de développer une application pour pouvoir manipuler ces fragments afin de les reconstituer. La reconstitution est un travail qui est fastidieux pour les archéologues. Avec une application de traitement d'images, ces reconstitutions seront simplifiées, à partir de l'instrument numérique.

L'objectif du projet est de **développer une Interface Web qui permettra aux archéologues de visualiser, de manipuler les fragments de Papyrus, ainsi que d'y appliquer des algorithmes de traitements d'images**. Ce site web recensera toutes les images de fragments de Papyrus ainsi que leurs méta-données. L'utilisateur devra avoir accès à plusieurs options de manipulation d'images ; c'est-à-dire retournement, rotation, changement d'échelle ou déplacement. À l'aide des algorithmes préalablement conçu par Antoine Pirrone, l'utilisateur pourra appliquer divers traitement et filtres sur l'image. De plus pour aider à assembler des fragments un algorithme est déjà à disposition. Celui-ci retourne les potentielles associations de fragments. Sur l'interface finale, les archéologues pourront les assembler convenablement, les traiter, les visualiser, eux mais aussi leurs méta-données. Enfin, les méta-données seront modifiables et enregistrables sur le serveur de l'application.

1.2 État de l'art

Pour ce projet, rien n'est imposé concernant l'ergonomie de l'interface et les choix techniques. Ces choix techniques possibles comprendront le ou les langages utilisés, la structure de l'interface et l'architecture client/serveur.

Exemples d'interfaces graphiques pour la visualisation et manipulation d'images :

Dans un premier temps, il peut exister des interfaces graphiques pour la visualisation et la manipulation d'images :

Il n'existe pas à notre connaissance, d'Interface Web disponible correspondant à la demande de notre client Cependant, il existe différentes Interfaces Web concernant le traitement d'images génériques.

Parmi ces interfaces de traitement d'images génériques il existe "G'MIC Online" [1]. Elle a été créé à partir d'un logiciel de traitement d'image écrit en C++. Elle a été développée par Le GREYC, Laboratoire de Recherche en Informatique, Image, Automatique et Instrumentation du CNRS établit à Caen. Elle permet de retoucher des images en appliquant des algorithmes issus d'une boîte à outils.

En revanche, il y a peu d'application dans le domaine de l'archéologie qui ont été développées, mais une Interface Web a été développée pour recenser tous les sites archéologiques existants en les situant sur une carte (MapBox). Ce site, "Archéosites" [2] n'est pas exclusivement destiné aux archéologues professionnels. Graphiquement, il est constitué d'une carte où des pictogrammes indiquent les sites archéologiques d'intérêt. Il est possible de zoomer, de déplacer la carte et de cliquer sur l'un des pictogrammes pour afficher les informations relatives au site choisi. De plus, sur le côté gauche, un menu interactif répertorie tous les endroits disponibles. Les développeurs ne semblent pas avoir communiqué sur le développement de leur interface ni de leur site. Cependant, le public peut participer au recensement de sites à l'aide de photos, de vidéos ou de textes explicatifs.

Outils de développement web existants :

Diverses outils de développement Web existent aujourd’hui :

Du côté client : Le développement web est dominé par le trio HTML5/CSS3/JavaScript : JavaScript est 6ème au TIOBE [3] index et 3ème au PYPL index [4]. L'avantage de ces technologies repose sur leur portabilité, leur inter-operabilité et leur compatibilité. De plus, la présence de framework JavaScript ou CSS tel que AngularJS [5] ou W3.CSS [6] permettent de grandement simplifier le déploiement de ces technologies web.

Du côté serveur : Plusieurs langages de programmation peuvent être utilisés pour communiquer avec le serveur. On trouve notamment PHP côté 8ème au TIOBE index et 5ème au PYPL index.

Plus récemment, en 2009, est apparue une nouvelle technologie open source : NodeJS [7]. Elle permet de mettre en place un environnement Javascript côté serveur. Il est basé sur le moteur d'exécution de Chrome : V8 Engine. La popularité de NodeJS s'est établit sur sa rapidité d'exécution notamment grâce à la compilation JIT et à son modèle non-bloquant.

1.3 Analyses des besoins fonctionnels et non fonctionnels

1.3.1 Besoins Fonctionnels

Dans un premier temps, une liste des besoins fonctionnels est établie :

Affichage d'un menu permettant le choix des fragments à afficher :

L'interface doit comporter un menu dynamique proposant d'afficher l'ensemble des papyrus. La référence du fragment est accompagnée de sa miniature. Cette miniature doit pouvoir être drag and copy dans le canvas. Le menu comporte des accordéons permettant de choisir entre les différentes déclinaisons du fragment.

Affichage de plusieurs images simultanément :

Plusieurs fragments de papyrus peuvent coexister dans le canvas.

Manipulation des images par l'utilisateur :

L'utilisateur doit avoir la possibilité de manipuler les images afin de mieux les visualiser. Les options proposées sont : rotation, zoom IN/OUT, déplacement de l'image et suppression d'une ou des images. Ces options doivent permettre une manipulation des images de façon indépendante. Cependant afin de permettre aux archéologues d'assembler plus facilement les fragments d'intérêts, certaines manipulations devront être applicables à tous les fragments présent sur le canvas. Ainsi une manipulation consistant à retourner tous les fragments simultanément, c'est à dire changer de face (Recto ou verso) devra être créée.

Affichage des méta-données associées aux différents papyrus :

Les méta-données de chaque papyrus doivent pouvoir être visualisées par l'utilisateur lorsqu'il sélectionne une image. Ainsi l'interface devra permettre d'afficher pour chaque fragment son langage d'écriture, sa taille réelle, s'il représente une bordure ou non ainsi que la face qu'il représente associée à son type de numérisation (recto ou verso, infrarouge ou couleur).

Assemblage et de-assemblage de fragments :

L'interface devra proposer à l'utilisateur d'assembler des fragments entre eux et de récupérer cette assemblage sous forme d'une seule image. Cet assemblage devra ensuite pouvoir être re-décomposer afin de récupérer individuellement chaque fragment qui le compose.

Modification des métadonnées par l'utilisateur :

Mise en place d'un système sous forme de formulaire permettant à l'utilisateur d'entrer et/ou modifier les métadonnées associées à une image et de les enregistrer.

Création d'un projet :

L'interface devra proposer à l'utilisateur un espace personnel de création de projet sécurisé lui permettant d'y importer des images et de les stocker y compris les assemblages qu'il aura réalisé.

Portabilité, Compatibilité et Interopérabilité de l'application :

Portabilité et Compatibilité : L'interface et les fonctionnalités associées à l'application doivent être totalement compatibles avec un maximum de navigateurs différents sur des systèmes d'exploitations différents. De plus, elle doit être suffisamment souple et robuste pour que toutes les versions soient compatibles. L'application doit également pouvoir fonctionner sur des appareils avec des tailles d'écran différentes et elle doit embarquer des fonctionnalités tactiles.

Interopérabilité : l'ensemble des technologies utilisées doivent fonctionner ensemble de manière optimale.

Création d'un serveur :

Création d'un serveur NodeJS mettant à disposition l'interface. Le serveur stocke les images de fragments, les fichiers XML ainsi que les algorithmes de traitements d'images.

Exécution des scripts python :

L'utilisateur doit pouvoir lancer les algorithmes, qui s'exécutent côté serveur, depuis l'interface et afficher le résultat dans le canvas.

Intégration de nouveaux algorithmes et de nouvelles fonctionnalités :

L'intégration de nouveaux algorithmes et de nouvelles fonctionnalités doit être simple, efficace et ne doit pas présenter d'instabilités si celle-ci suit le modèle d'intégration du manuel utilisateur.

1.3.2 Besoins non Fonctionnels

Dans un deuxième temps, une liste des besoins non fonctionnels est aussi réalisée. Les besoins non fonctionnels listés ci-dessous sont nécessaires pour permettre une bonne utilisation, rapide et efficace de l'interface par l'utilisateur.

Interface intuitive et ergonomique :

L'interface a été conçue pour être responsive.

Vitesse d'exécution :

L'application doit présenter de bonnes performances dans un soucis de confort pour l'utilisateur. Les latences doivent être réduites au maximum lors de l'utilisation des fonctionnalités de manipulation d'image.

Système d'utilisateurs et sécurité :

Concernant le Pentest et audit sécurité de l'application [8] [9] [10] [11] [12] [13], il s'agira de protéger un maximum le serveur et le client selon les exigences du client : l'accès en écriture doit être restreint (mots de passe ou clés de sécurité).

Ajout d'options d'accessibilités :

Retour visuel lorsque l'utilisateur sélectionne une image.

Enregistrement d'une image après traitement :

Possibilité pour l'utilisateur de visualiser la version traitée d'une image, obtenue après application des algorithmes et d'enregistrer cette version.

1.4 Choix de réalisation

Afin de répondre aux besoins listés précédemment, l'interface imaginée se présentera sous la forme d'une page web, comprenant différents onglets. Il y aurait notamment, un onglet pour la visualisation du canvas, un onglet pour la modification des méta-données, un onglet pour la création d'un répertoire et un onglet de sauvegarde d'assemblage. Il a notamment été imaginé pour l'onglet de visualisation une colonne sur la gauche où les images seraient disponibles sous forme d'accordéon avec possibilité d'accéder aux différentes versions des images (Recto, Verso, InfraRouge). Au milieu, apparaîtrait le canvas avec les images sélectionnées. Sur celles-ci, il y aurait alors possibilité d'y faire des manipulations à l'aide d'une boîte à outils.

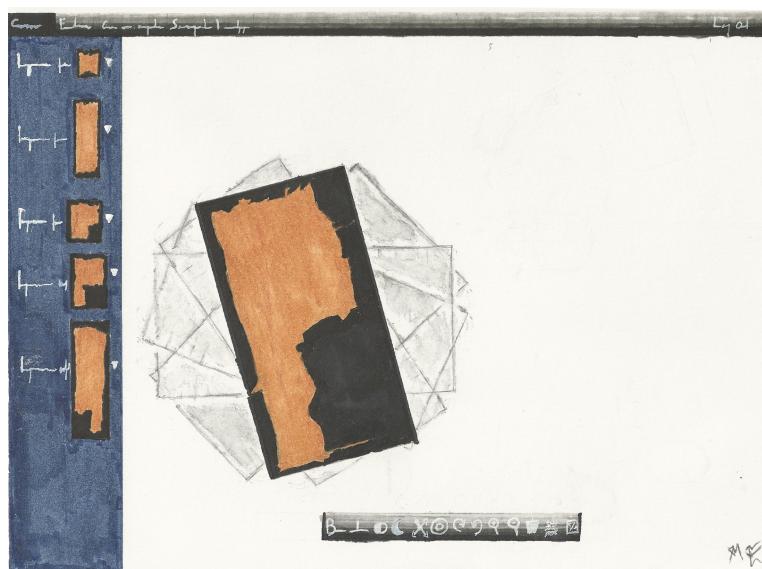


FIGURE 1.2 – Schéma de l'interface imaginée

Concernant les langages à utiliser, du côté client, ont été choisis :

- HTML5
- CSS3/framework W3.CSS
- JavaScript/framework AngularJS

HTML5 est la dernière version de HTML. Il s'agit du langage standard pour créer du contenu web.

Le framework W3.CSS sera utilisé pour décrire le style de l'application.

Les fonctionnalités relatives à la manipulation des images et à l'extraction des méta-données

seront implémentées en JavaScript. Le framework AngularJS servira à simplifier l'intégration de fonctionnalités JavaScript. En effet, l'utilisation du framework AngularJS permettra notamment, en détectant et surveillant les modifications sur un objet JavaScript, la synchronisation automatique des données entre le modèle et les composants de la vue.



FIGURE 1.3 – HTML5, W3CSS et AngularJS

Du côté serveur, NodeJS est utilisé pour créer le serveur de l'application. Il fournira tous les services assurant le bon fonctionnement de l'interface web comme la gestion des images (chargement et actualisation de la table de références) et la gestion des fichiers XML.

Côté serveur, le format JSON (JavaScript Object Notation) sera aussi utilisé, il permettra une bonne structuration des informations et facilitera l'échange des données. IL pourra notamment être utilisé pour la gestion des images, et la gestion de l'authentification des utilisateurs.



FIGURE 1.4 – JSON et NodeJS

Chapitre 2

Conception

L'interface proposée est une interface web qui fonctionne donc sous le modèle serveur-client. Il s'agira ici d'une architecture de type centralisée, toutes les données seront centralisées sur un seul serveur simplifiant les contrôles de sécurité et la mise à jour des données. L'accès à l'interface nécessitera une authentification de l'utilisateur via un identifiant et un mot de passe qui lui permettront d'accéder pleinement aux données mises à disposition. Le serveur sera lui configuré de manière à fonctionner en HTTPS.

2.1 Client

2.1.1 L'interface

Pour la création de l'interface, il s'agira de créer une page web sous protocole HTTPS en HTML5. Elle devra être divisée en 4 onglets. Le premier devra permettre de visualiser le contenu du canvas (images sélectionnées uniques, ou multiples, assemblées ou non) et permettre la manipulation de ces images.

Le deuxième onglet devra permettre de modifier les méta-données via un formulaire.

Le troisième onglet devra créer un répertoire de travail permettant l'enregistrement des assemblages réalisés.

Enfin, le quatrième onglet servira de sauvegarde de l'assemblage ou de suppression de cet assemblage.

Dans un premier temps, le modèle de l'interface sera créé côté client en HTML5. De plus, pour gérer le style il faudra implémenter des feuilles de style en CSS.

En ce qui concerne les manipulations de fragments, il s'agira d'implémenter côté client (ou serveur quelques fois) des scripts JavaScript pour l'interaction avec la page. Deux scripts devront être implantés, l'un permettra de gérer les images hors du canvas avant de les importer (dimensions réelles, ratio entre les fragments et importation), l'autre sera attaché à la création du canvas et des manipulations possibles au sein de celui-ci.

2.1.2 Le Canvas

L'onglet Canvas représentera la base principale de l'interface. Dans un premier temps, il faudra récupérer les images au format PNG ainsi que les données au format XML. Il y a pour chaque fragment, 4 images différentes : une recto et une verso dans leurs versions couleurs, et une recto et verso en InfraRouge. Les méta-données concernant les fragments seront contenues

dans un fichier XML, pour chacun des fragments. Les références et les sources des différentes images disponibles pour une session donnée devront être stockées côté serveur, pour chaque fragment. Plusieurs fonctionnalités devront apparaître dans cet onglet canvas : le canvas en lui-même où seront manipulées les images ainsi qu'un panneau latéral contenant les miniatures des fragments disponibles dans la session sur laquelle l'utilisateur est connecté, lesquels pourront être importés dans le canvas.

L'utilisateur aura la possibilité d'afficher n'importe quelle version du fragment que l'on veut importer dans le canvas (recto ou verso, couleur ou InfraRouge) à partir du panneau latéral. Pour importer la version recto-couleur dans le canvas, il faudra glisser et déposer la miniature dans le canvas (modèle drag and drop) afin de l'importer dans celui-ci. Il faut également pouvoir visualiser plusieurs fragments en même temps.

Pour importer une autre version (qui pourra être modifiée par la suite), cliquer sur une miniature permet de dérouler un menu présentant les différentes versions du fragment correspondant. L'utilisateur n'aura alors qu'à cliquer sur la version souhaitée. Afin de respecter le ratio entre les tailles des différents fragments affichés dans le canvas, il faudra pouvoir récupérer les métadonnées associées aux dimensions du fragment que l'utilisateur veut afficher et les faire entrer en jeu lors de l'importation des images dans le canvas (voir partie serveur).

Une barre d'outils devra également être incorporée au canvas pour permettre à l'utilisateur de manipuler les images des fragments affichés. Sur cette barre d'outils seront placés des boutons. Ils serviront aux différentes manipulations comme changer le mode de visualisation (recto, verso, InfraRouge) de l'ensemble des fragments affichés, faire pivoter ou zoomer/dé-zoomer une image, supprimer une image du canvas ou bien, lancer les scripts python incorporés.

L'un des objectifs principaux est l'assemblage des fragments à partir d'un algorithme en Python « best matches » qui retourne une matrice de score de match entre chaque fragment. Cet algorithme sera exécuté à l'initialisation du serveur et il faudra créer un bouton pour traiter cette matrice et retourner les fragments présentant les meilleurs scores avec celui sélectionné. Il y a également un autre algorithme, toujours en Python, qui permet de faire un threshold sur l'image, il faudra donc créer un autre bouton qui permettra de lancer ce script sur l'image sélectionnée et d'incorporer l'image obtenue aux images disponibles pour la session. Ces deux scripts Python seront exécutés côté serveur.

L'utilisateur pourra également être amené à vouloir visualiser les méta-données associées à un fragment en particulier. Ces informations sont stockées dans des fichiers XML propres à chaque fragment. Il faudra alors y récupérer certaines données comme le bord, la taille, la langue et la localisation du fragment en question afin de les afficher dans une fenêtre.

2.1.3 Le Formulaire

Un onglet formulaire doit être créé, celui-ci permettra à l'utilisateur de modifier les métadonnées liées à un papyrus. Pour cela, il faudra implémenter sur l'interface html et sur un fichier JavaScript au niveau du serveur, de quoi modifier les données. L'utilisateur pourra modifier diverses informations liées à l'image qu'il traite, telle que la taille, l'origine ou encore rajouter des commentaires. Le contenu de ce formulaire sera envoyé au serveur et le fichier XML correspondant sera modifié côté Serveur.

2.1.4 Creation de repertoire

Un onglet répertoire devra être créé afin de travailler sur sa propre session à l'aide d'identifiants que l'utilisateur choisira. Cela créera un utilisateur/projet. Avec cette session, l'utilisateur pourra alors accéder à toutes les fonctionnalités du système.

teur pourra récupérer les manipulations, les assemblages ou les traitements qu'il aura effectué la fois précédente. Cette fonctionnalité se déroulera côté Serveur, sur un fichier JavaScript. Il sera aussi possible de supprimer un répertoire.

2.1.5 Sauvegarde d'un assemblage

Un onglet de sauvegarde de l'assemblage devra être créé afin de sauvegarder, sous la forme d'une entité unique, l'assemblage créé dans le canvas à partir de plusieurs images de fragments et de l'envoyer au serveur pour actualiser les images disponibles dans la session. Pour désassembler cette entité, une fonction sera implémentée dans la barre d'outils, le serveur se chargera d'actualiser les données en conséquence et de retourner les images à l'origine de l'assemblage désassemblé.

2.2 Serveur

2.2.1 Gestion des données

Import et initialisation des données

Afin de permettre à l'utilisateur d'avoir pleinement accès aux données, celles-ci doivent être correctement intégrées et gérées par l'interface. Les données brutes sont implémentées dans un répertoire côté client, mais seront initialisées et traitées côté serveur.

Afin de gérer au mieux les données, celles-ci sont initialisées de façon à obtenir une table complète de toutes les données papyrus présentes sur le serveur. Cette table sera implantée au format JSON selon un modèle où pour un élément (un fragment) sont associées toutes les données qui lui sont référencées (référence, sources de tous les types d'images, ses métadonnées sous format xml et la session associée).

Au cours des manipulations diverses de l'utilisateur cette table est susceptible d'être modifiée, des données peuvent être ajoutées notamment si l'utilisateur souhaite ajouter un assemblage ou effectuer un threshold sur une image. Il s'agira donc d'inclure dans un des scripts principaux du serveur des fonctions permettant la modification et l'actualisation de cette table.

Gestion de l'assemblage-désassemblage des fragments

L'utilisateur à la possibilité de créer ou détruire un assemblage via un onglet de l'interface. Cette fonctionnalité sera gérée côté serveur. Lorsque l'utilisateur souhaite enregistrer un assemblage, il génère côté client un aperçu qui pourra être enregistré et implanté dans le canvas. L'image de l'assemblage est alors envoyée au serveur qui se chargera de l'ajouter à la table d'images et d'actualiser celle-ci.

Le désassemblage est aussi géré côté serveur, lorsque l'utilisateur souhaite supprimer l'assemblage, le serveur sera alors chargé de supprimer la référence de l'image dans la table, de purger le canvas est de mettre à jour la table des papyrus.

2.2.2 Intégration et exécution des scripts

L'utilisateur a la possibilité d'effectuer un Treshold sur l'image de son intérêt et d'afficher les différents scores de compatibilité entre les fragments. Ces manipulations sont rendues possibles grâce à l'exécution de deux scripts python côté serveur.

L'utilisateur indiquera côté client via un bouton son souhait de traiter l'image. Le serveur traite alors la requête en récupérant les informations envoyées par le client, exécute l'algorithme en question et envoi alors la réponse sous différentes formes au client. Ainsi dans le cas de l'algorithme Threshold, le client envoie sa requête, le serveur devra alors exécuter le script associé, et renvoie alors au client l'affichage de l'image d'intérêt traitée. Lorsque l'utilisateur souhaitera obtenir la liste des scores d'assemblages d'un fragment avec d'autres, il envoie sa requête au serveur, qui exécute le script de score associé, ce script renvoi alors une table de scores côté client qui se chargera d'afficher via l'interface la liste décroissante des fragments et leur score associé.

Le serveur doit donc contenir des scripts principaux moteurs qui permettront de récupérer les chemins relatifs aux différents scripts, les exécuter et renvoyer une réponse au client qu'elle soit sous la forme d'une image visualisable ou d'une liste de fragments triés par score d'assemblage.

2.2.3 Gestion de la sécurité et des sessions personnelles

Plusieurs aspects de sécurité seront traités. L'application devra par exemple être robuste aux attaques de type cross-site-scripting (XSS), cross-site request forgery (CSRF), MitM ainsi qu'au sniffing. Le serveur n'autorisera aussi que les connections en HTTPS. Les certificats de sécurité utilisés pour HTTPS seront générés à l'aide d' ECDSA avec la courbe elliptique secp256k1. Les recommandations cryptographiques du *Référentiel Général de Sécurité version 2.0 Annexe B1 Mécanismes cryptographiques Règles et recommandations concernant le choix et le dimensionnement des mécanismes cryptographiques Version 2.03 du 21 février 2014* seront respectés.

2.3 Architecture établie

L'Architecture des programmes permettant la réalisation de l'interface (figure ??) est donc divisée en deux parties, une partie client et une partie serveur qui communiqueront entre elles sous un modèle de requêtes client exécutées et renvoyées par le serveur, on parle d'une architecture client-serveur à deux niveaux.

On retrouve ainsi côté client l'ensemble des fichiers et programmes définissants le graphisme de l'interface (fichiers .css), mais aussi le répertoire contenant l'ensemble des données qui seront mises à disposition (tous les types d'images associées à un fragment), les métadonnées associées mais aussi les matrices de scores d'association entre les différents fragments.

On retrouve aussi coté client les fichiers dits «moteurs» de l'interface, en Javascript, programmes qui permettront à l'utilisateur d'interagir avec l'interface et d'effectuer les manipulations sur les images de fragments et communiquer avec le serveur.

Enfin tous ces fichiers seront notamment appelés par le programme principal de l'interface permettant la conception même de l'interface utilisateur, le fichier *Interface.html*. L'ensemble de ces fichiers est inclus dans un répertoire dénommé *Secure*, et ils sont accessibles côté serveur,

uniquement suite à l'authentification de l'utilisateur.

Le serveur sera écrit et configuré via Node.JS (*ServerExpress.js* et *Route.js*). Il communiquera avec le client par requête http (GET/POST) pour l'envoie et la réception de données nécessaires à certaines fonctionnalités de l'interface (récupérer les méta-données dans les fichiers XML stockés sur le serveur, actualisation de la table contenant les images disponibles pour une session, récupération des *matches scores* ect). On retrouve également, toujours coté serveur, les fichiers permettant la mise en place du système de sécurité par authentification de l'utilisateur, ainsi que la création de session personnelle elles aussi gérées par les fichiers principaux *ServerExpress.js* et *Route.js*.

Chapitre 3

Réalisation

Les éléments définis dans le cahier des charges ont pu être réalisés selon le modèle expliqué dans ce chapitre.

Les résultats obtenus et fournis dans cette partie ont été générés grâce à l'utilisation d'un jeu de données fictif fourni par le client. Ces données sont similaires dans leur forme aux données réelles à disposition des archéologues, seulement le jeu de données fourni est grandement allégé.

3.1 Initialisation du Serveur

L'exécution du script *RunServer.sh* initialise le serveur. Le script vérifie l'absence de sessions antérieures et les supprime le cas échéant. Il lit ensuite les certificats qui seront utilisés. Après cela, NodeJS initialise ces dépendances puis les paramètre. Juste avant de lancer le serveur d'écoute à proprement parlé, il initialise les routes. Les routes sont l'ensemble des méthodes et des chemins utilisés lors de la communication Client/Serveur. Les chemins à suivre sont tous stockés dans des variables différentes pour plus de modularité. Le serveur n'autorise que les méthodes GET et POST ce qui le protège du Verb Tampering.

Une fois le serveur initialisé l'utilisateur peut accéder à la page *index.html* et peut s'identifier.

3.2 Accès à l'interface et authentification de l'utilisateur

3.2.1 Authentification

Dans le fichier *passwds.json* se trouve un objet JSON contenant les différents identifiants et mots de passe hashés.

L'utilisateur entre son identifiant et son mot de passe sur une page d'authentification, Figure 3.1. Dans *Route.js*, on teste si l'authentification est réussie. Pour cela on compare l'identifiant entré avec les différents identifiants présents dans *passwds.json*. Si l'identifiant y est présent, on compare alors le mot de passe entré par l'utilisateur avec les mots de passes présents dans le même fichier. Si l'authentification est effectivement réussie, l'utilisateur est redirigé sur la page de l'interface *interface.html*.

Dans le cas contraire, le serveur renvoie le code de statut de réponse HTTP "400 Bad Request" empêchant alors l'utilisateur d'accéder à l'interface.

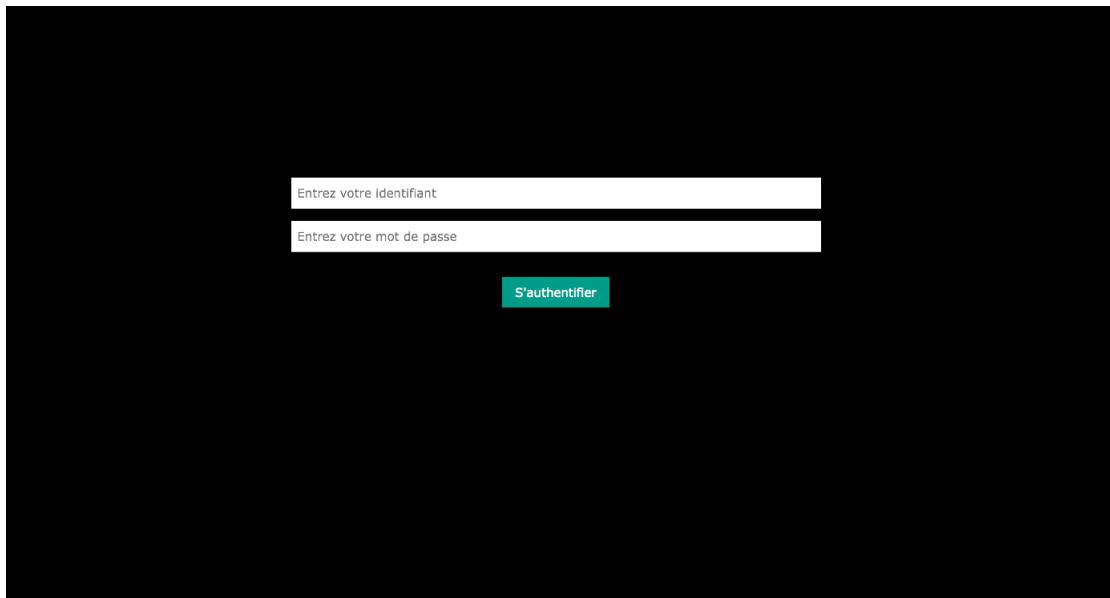


FIGURE 3.1 – Capture de la page d’authentification de l’interface

3.2.2 Ajout d’utilisateurs

Si l’utilisateur ne s’est jamais connecté à l’interface, il est possible d’y accéder en entrant la chaîne de caractères "main" dans les champs d’authentification. Il aura alors accès à l’interface et pourra créer une session personnelle à partir de l’onglet "Créer un répertoire". (voir section 3.5)

3.3 Création et fonctionnement général du menu Canvas

Le canvas est construit sous la forme d’un objet appelé Area. une fonction start() invoque la méthode start() de l’object Area.

3.3.1 Area

Cette méthode crée un élément <canvas> et l’insère dans l’élément <div id="CanvasHolder"> du body. Area contient toutes les variables relative au canvas.

Les propriétés width,height,top et left sont définies selon les dimensions du "CanvasHolder".

Area est donc définie par sa propriété .canvas mais également par son centre (Area.x/Area.y), son mode ("zoom"/"rotate"), son échelle (Area.scale), son contenu (Area.images) sous la forme d’un tableau contenant des objets que nous détaillerons dans la section suivante ainsi que par son contexte. Area comporte également des variables globales qui seront utilisées par les fonctions mises en place par les *event listeners*. L’état de Area est actualisé toutes les 30ms grâce à une fonction nommée updateArea().

3.3.2 Area.images

Area.images contient les images présentes dans le canvas, sous forme d'objet : Chaque objet est accessible par son index (Area.images[0] par exemple) et est défini par les propriétés suivantes : sa source Area.images[i].src, sa référence (.ref), sa position (.x et .y) et son angle de rotation (.angle). Ces propriétés sont initialisées lors de la création de l'objet image (Area.images[i].image = new Image()) et son chargement, le tout dans la fonction component(src,ref). Ces propriétés seront utilisées par les méthodes de Area.context : translate(), scale(), rotate(), drawImage() ect pour, dans un premier temps afficher l'image dans le canvas, puis actualiser cette image. Afin de ne pas altérer les autres images dans le canvas, la méthode restore() est utilisée. Pour ajouter une image au canvas, il est donc nécessaire d'injecter l'objet correspondant dans Area.images.

3.4 Importation des données

Une fois l'utilisateur authentifié, les dossiers relatifs au répertoire Secure sont alors accessibles, l'interface, Figure 3.2, définie par le fichier *interface.html* et ces CSS associés sont exécutés. Tout ceci se déroule dans la partie Client.

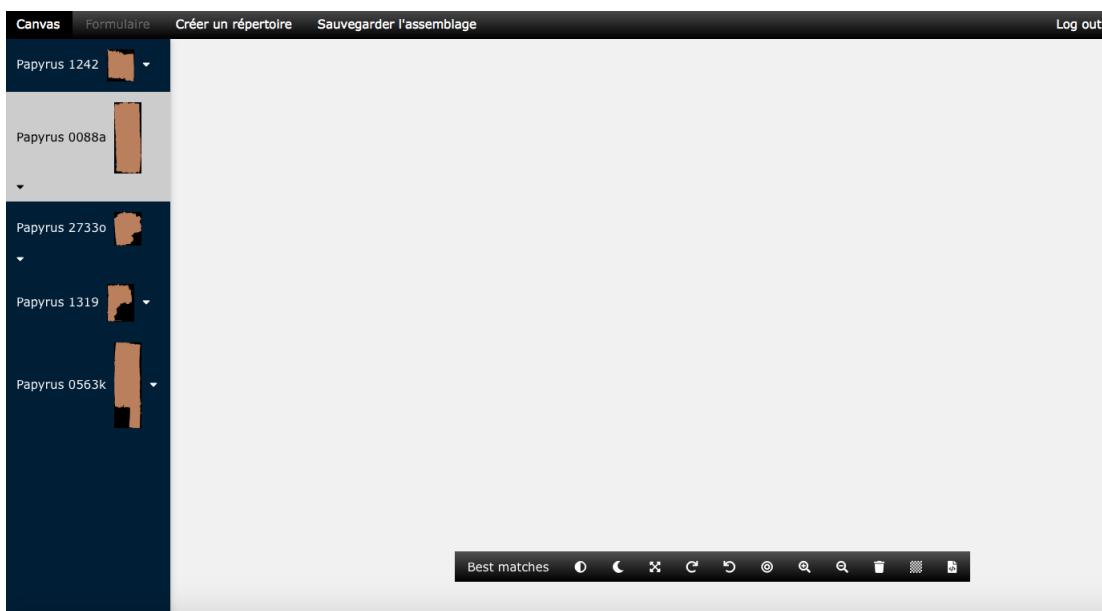


FIGURE 3.2 – Capture de l'interface

3.4.1 Crédit et remplissage du panneau latéral

Afin de remplir le panneau latéral à partir de la table contenant les images utilisables dans la session, la directive angularJS "ng-repeat" définie dans le contrôleur "RepeatPapyrus" est utilisée.

La directive ng-repeat répète un ensemble de HTML, un nombre de fois donné. Dans le cadre de cette application, ce nombre de fois va dépendre du nombre d'éléments dans l'objet JSON obtenu via une requête http envoyée au serveur (objet : \$rootScope.papyrus disponible dans

toute l'application angular et servira à chaque fois que l'utilisateur déclenchera une action nécessitant cet objet). La directive va alors créer un nombre de div correspondant au nombre d'images disponibles dans le répertoire de travail. Les div obtenues sont remplies grâce aux expression d'AngularJS qui permettent d'insérer les propriétés de l'objet utilisé.

Elles contiennent alors une miniature dont la source correspond à celle de la version couleur de chaque fragment (item.RCL) et ont pour attribut id, la référence du fragment correspondant associée au suffixe _thb (item.THB) qui serviront à afficher les images dans le canvas (voir section 3.4)

3.5 Affichage des fragments dans le canvas

3.5.1 Drag and Drop

L'importation des images utilisables pour une session donnée peut se faire par la méthode du *drag and drop*. Pour rendre les miniatures présentes dans le panneau latéral déplaçables, on peut utiliser l'attribut **draggable** et mettre sa valeur à **True**. Une fonction "drag(event)" est associée à l'évènement ondragstart qui correspond à l'action de déplacer un élément avec la souris. Cette fonction permet de récupérer la valeur de l'attribut id qui contient la référence du fragment suivi du suffixe "_thb" de l'élément déplacé grâce à l'objet dataTransfert et la méthode setData(). Cette dernière permet de sauvegarder une chaîne de caractères qui sera transmise à l'élément HTML qui accueillera l'élément déplacé. Cette méthode prend deux arguments : **event.dataTransfer.setData("text/html", event.target.id);**

Par défaut, le navigateur peut interdire de déposer un quelconque élément où que ce soit dans la page Web, donc une deuxième fonction, allowDrop(event), a été associée par précaution à l'évènement ondragover afin d'annuler ce comportement par défaut : **event.preventDefault();**

Enfin, une troisième fonction, drop(event) associée à l'évènement **ondrop** utilise la méthode **getData** de l'objet **dataTransfert** pour récupérer l'attribut Id de la miniature déplacée. À partir de cet attribut, la source de l'image pourra être récupérée en utilisant cet Id pour rechercher l'élément dans la page (getElementById()); et en récupérer la valeur de l'attribut src. La référence de l'image est obtenue en supprimant le suffixe "_thb" et grâce à la valeur de l'Id récupéré dans la fonction drag(). Un nouvel objet image sera alors créé à partir de cette référence et la source de l'image. Cet objet sera injecté dans le tableau Area.images.

Exemple : **Area.images.push(new component(elem.src,newId));**

3.5.2 Modification de l'affichage d'un fragment depuis la barre latérale

Il existe également une autre possibilité pour modifier l'affichage d'une image de fragment depuis le panneau latéral, Figure 3.3. Un clic droit sur la référence d'un fragment permet de déclencher un *dropdown* via la directive ng-click de AngularJS reliée à la fonction AccFunc(id) définie dans le contrôleur RepeatPapyrus (défini dans Crtl.js).

Cette fonction prend l'id de l'élément cliqué en argument et rajoute à l'attribut classe la valeur "w3-show", si elle n'est pas déjà présente et le menu s'ouvre. Dans le cas contraire, cette valeur est enlevée et le menu se ferme.

Ce menu déroulant est composé de quatre éléments d'ancre Html (<a>) possédant chacun

une directive ng-click les reliant à la fonction ChangeAttr(ref,src) définie dans le contrôleur PictCrtl. Cette fonction teste si la référence de la version de l'image est déjà présente parmi les objets stockés dans Area.images et crée un objet. Celui-ci est injecté dans ce tableau avec la source associée à l'élément cliqué et la référence du fragment (valeur de l'attribut Id de l'élément cliqué) si la référence n'est pas trouvée.

Si la référence est trouvée, alors la source de l'image présente dans le canvas sera remplacée par la source associée de l'élément cliqué afin de modifier la version du fragment en question dans le canvas.

Nb : Cette solution est la seule option possible pour les appareils tactiles sans souris ni *touchpad*.

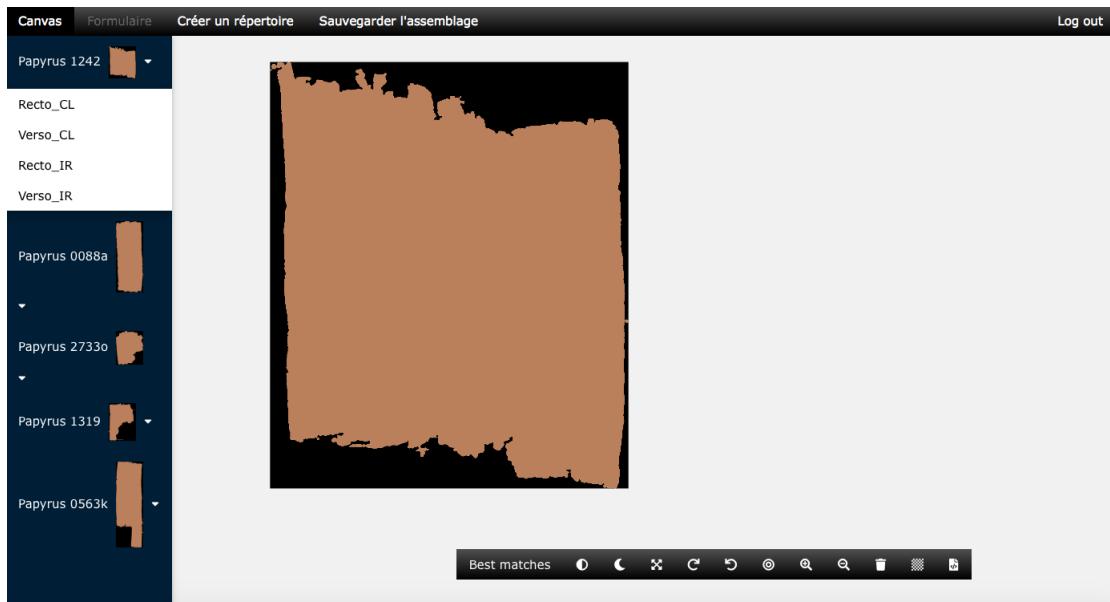


FIGURE 3.3 – Visuel de l'interface après sélection d'une image

3.5.3 Mise à l'échelle réelle des images

Les dimensions des différentes images affichées dans le canvas ainsi que le ratio largeur/hauteur doivent être conformes aux dimensions réelles des fragments correspondants. Pour cela, la fonction getRealSize(ref) (*ManagePict.js*) prend la référence d'une image en argument et modifie les propriétés width et height du dernier élément de Area.images, qui est l'élément que l'utilisateur souhaite ajouter, en sortie.

Pour ce faire, le client va, dans un premier temps envoyer une requête http (méthode GET) pour récupérer le contenu de la table également utilisée pour remplir le panneau latéral sur le serveur. Le serveur retourne un objet dans lequel se trouve la référence des fichiers xml associés aux différents fragments. Cette référence est récupérable grâce à la référence du fragment passée en argument.

La fonction procède ensuite à l'envoie d'une seconde requête vers l'emplacement du fichier XML correspondant au fragment.

Si la requête s'est bien déroulée, on peut récupérer les dimensions réelles du fragment grâce à la fonction turnRealSize(xml) qui prend la réponse retournée par la requête en argument et va chercher les informations des dimensions. Les dimensions sont définis à travers l'attribut "unit" et les balises "width" et "height".

Le principe reste le même pour toutes les autres fonctionnalités nécessitant d'accéder aux métadonnées. La seule différence est celle que l'on peu observer entre les informations concernant l'unité et celles sur les dimensions et dépend de la structure des fichiers au niveau de ces informations.

3.6 Manipulation sur les fragments

3.6.1 Déplacement des images dans le canvas

Le déplacement des images dans le canvas est géré par trois fonction pour chaque type d'appareil (tactile/non tactile). Ces fonctions se déclenchent suite à des évènements.

Dans un premier temps, une première fonction se déclenche quand l'utilisateur presse la souris ou pose son doigt sur la partie canvas de l'écran. Cette fonction teste si l'action a lieu dans la zone occupée par une image en utilisant une méthode définie dans Area qui est : Area.images[i].contains(mousex,mousey); qui itère dans Area.images et compare les coordonnées x et y de la souris avec les valeurs des extrémités x et y de Area.images[i] de façon à s'adapter au niveau de zoom. Si les coordonnées x et y sont toutes deux comprises entre les minimales et maximales autour du centre, alors la fonction retourne la valeur **true**. La fonction principale modifie alors la valeur de la propriété Area.selection = Area.images[i] et celle de la variable globale touched = **true** autorisant alors la fonction déclenchée par l'évènement mousemove (ou touchmove) à modifier la valeur du centre de l'image (Area.selection.x et Area.selection.y). Ces deux valeurs sont modifiées en leur ajoutant ou soustrayant la valeur de la différence entre les coordonnées de la souris et les anciennes coordonnées du centre de l'image. La méthode Area.images[i].translate() appelée toutes les 30ms dans update permet de replacer l'image à ses nouvelles coordonnées.

La fonction s'assure également que l'image ne sorte pas des limites du canvas en comparant Area.selection.x et Area.selection.y avec les coordonnées des extrémités du canvas.

Enfin, la troisième fonction s'active lors des évènements lorsque l'utilisateur relâche le clic de la souris ou retire ses doigts de l'écran (mouseup/touchend). La variable globale touched est réglé sur **false**, empêchant alors la modification des coordonnées de l'image.

3.6.2 Autres fonctionnalités : Barre d'outils



FIGURE 3.4 – Capture de la Barre d'Outils disponible sur l'interface

D'autres fonctionnalités sont disponibles à partir d'une barre à outils, Figure 3.4, disposée en bas de l'interface. Les fonctions derrière ces fonctionnalités sont attachées aux éléments <button> et sont déclenchées quand l'utilisateur effectue un clic gauche sur l'un d'eux via la directive AngularJS **ng-click** reliée au contrôleur "ToolsCommand" dans le fichier *Crtl.js*.

Premièrement, quatre boutons sont très utiles dans la manipulation des fragments. Il s'agit des boutons de rotation sur la droite et sur la gauche et des boutons de Zoom en avant (+) ou de Zoom en Arrière (-). Pour la Rotation il s'agit d'augmenter, pour la Droite, la propriété *Area.selection.angle* ou bien de réduire cette propriété pour la rotation Gauche. Cette propriété est utilisée par la méthode **Area.images[i].rotate**

Pour le Zoom, il s'agit ici d'augmenter la valeur de la propriété de l'*Area.scale* qui est utilisée

pour redimensionner *Area*, ou bien, pour Dé-zoomer, de réduire la valeur de cette propriété. Ces fonctionnalités sont aussi déclenchables par l'intermédiaire des touches fléchées du clavier ou de la molette de la souris dans le cas du zoom à l'aide de la propriété **deltaY** de l'évènement "**wheel**" dont la valeur est supérieure à 0 lorsque l'on scroll vers le bas (*Area.scale* augmente si *deltaY* > 0).

Pour effectuer une rotation à la souris, il faut effectuer un double clique gauche sur une image afin de la sélectionner. La propriété *Area.mode* prend alors la valeur "rotate". Les coordonnées du centre de l'image ne sont plus modifiées et *Area.selection.angle* est calculé à partir de l'angle entre le centre de l'image (*Area.selection.x/y*) et les coordonnées de la souris. Ré-effectuer un double clique gauche n'importe où dans le canvas permet de revenir au mode **Area.mode = "move"**; Les fonctions seront écrites dans le fichier *ManageCanvas.js*.

Ensuite les autres boutons disponibles correspondent au bouton de Suppression, au bouton de Désassemblage, au bouton pour passer en Mode Nuit/Jour et au bouton pour Changer le Mode Visualisation.

Le bouton de Suppression retire du canvas une image sélectionnée par l'utilisateur en utilisant la méthode **Area.selection.remove()** si *Area.selection*!== null;

Le bouton de Désassemblage permet de désassembler des images précédemment assemblées. La fonction *DisassCompound()* appelle la méthode **Area.selection.disass()** dans *ManageCanvas.js*. Elle vérifie que l'image sélectionnée est bien présente dans **Area.images**. Si elle l'est, la référence de l'image est stockée dans une variable. La fonction vérifie aussi si la référence comporte bien la chaîne de caractère "Compound". Une requête http/GET est envoyé au serveur pour accéder au contenu de la table de références ainsi qu'à l'objet JSON correspondant à l'assemblage et regroupant les propriétés des objets images utilisées pour l'assemblage.

Par itération, les différentes informations (ref, src,x,y,width,height,angle) sont récupérées dans cet objet. La fonction s'occupe ensuite d'injecter les objets images dans *Area.images* et de restaurer les valeurs de pré-assemblage pour toutes les propriétés de l'image.

Enfin, les données concernant l'assemblage sont envoyées vers le serveur pour y être détruites. Le bouton du Mode Nuit, permet de rajouter un fond sombre sur le fond du canvas en modifiant la propriété style du "canvas". La fonction bascule la propriété "mode" de *Area.canvas* entre "light" et "night".

Le bouton du Changement de Mode de Visualisation permet de choisir quelle image de fragment l'utilisateur souhaite, que ce soit le recto ou le verso de la version couleur ou le recto et le verso de la version infrarouge. La fonction *changeVisual()* est associée aux éléments du menu déroulant par la directive "**ng-click**", menu lui-même associé au bouton de la barre de fonctionnalités. Cette fonction est définie dans le contrôleur angularJS "ToolsCommand". Elle agit en parcourant *Area.images* et en modifiant les 8 derniers caractères de la source des images par le suffixe correspondant au visuel choisi par l'utilisateur.

Pour les images issues de Treshold, on ne change pas la source car il n'existe qu'une seule version.

3.6.3 Application des algorithmes de Treshold et de Score d'assemblage

Treshold

Une requête est envoyée au serveur avec la méthode POST vers l'emplacement du script python correspondant. La source et la référence de l'image sélectionnée y sont envoyées. Ces informations servent à récupérer l'image à traiter.

Sur le serveur est créé un environnement python, grâce à une dépendance dédiée à cet effet, permettant ainsi d'exécuter le script. Ce script prend en entrée le chemin de l'image à traiter ainsi que le chemin de destination de l'image obtenue après traitement. Une fois l'image traitée, elle est rognée puis enregistrée dans le serveur. Les tables de projets et de papyrus sont ensuite actualisées. Le fichier de métadonnées inscrit dans la table papyrus est celui de l'image non traitée.

L'image obtenue pourra ensuite être affichée selon les méthodes vu précédemment (voir section 3).

Scores d'assemblage

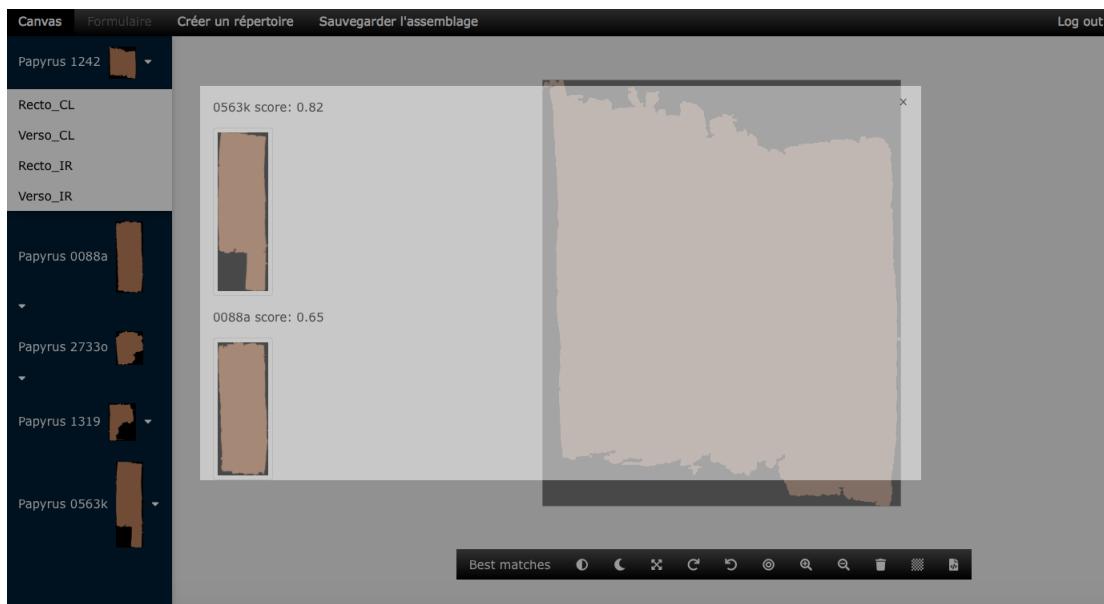


FIGURE 3.5 – Résultat de l'application de l'algorithme de scores d'assemblage possibles entre fragments

Le script python fourni par le client, n'est pas utilisé en l'état. Le but est de comparer chaque fragment avec tous les autres. La création d'une matrice de score sous forme d'objet JSON a été choisi pour répondre le plus efficacement possible à ce besoin. De ce fait la fonction implémentée a pour but de parcourir cette matrice et d'en ressortir les scores qui intéresseraient l'utilisateur. Les miniatures des images ayant les meilleurs scores (seuil= 0.60) sont affichés dans un menu modal 3.5.

La matrice de score est récupérée à l'aide d'une requête (GET) à l'url `"/secure/Datas/scoreMatrix.json"`. La fonction va parcourir le contenu de la matrice à la recherche d'une référence identique à l'image sélectionnée. Si elle la trouve, la fonction va stocker les références et les scores de matches correspondants dans un objet.

Les données contenues dans cet objet sont ensuite stockées dans un tableau de structure `[[ref,score],[ref,score],[.....]]` qui est trié par ordre décroissant de valeur de score. Enfin le menu modal est construit dynamiquement et affiché à l'écran. L'utilisateur pourra donc choisir d'afficher l'image qu'il souhaite.

3.6.4 Affichage des méta-données

La récupération du contenu du fichier XML correspondant au fragment sélectionné fonctionne comme dans le cas de la récupération des dimensions réelles d'un fragment (voir sous-section 3.4.3). En l'état présent, les informations récupérées et affichées sont les informations concernant les dimensions et la langue des écritures présentes sur le fragment.



FIGURE 3.6 – Affichage des méta-données associées à un fragment

3.7 Crédit d'un répertoire de travail, enregistrement d'assemblages et manipulation des fichiers disponibles pour la session

3.7.1 Crédit d'un répertoire de travail

Si l'utilisateur ne s'est jamais connecté à l'interface, il est possible de s'y connecter en tant que "main". Il aura alors accès à l'interface et pourra créer une session personnelle à partir de l'onglet "Créer un répertoire", Figure 3.7. Une fois l'identifiant et le mot de passe du nouvel utilisateur rentrés dans le formulaire, ils sont envoyés au serveur (requête https avec méthode de type GET).

Une fois que le serveur reçoit les identifiants, il vérifie qu'il ne sont pas déjà existant. Si c'est bon il actualise le fichiers de projets et de mots de passes. Les mots de passe sont chiffrés avec Argon2i (voir section Protections des mots de passe).

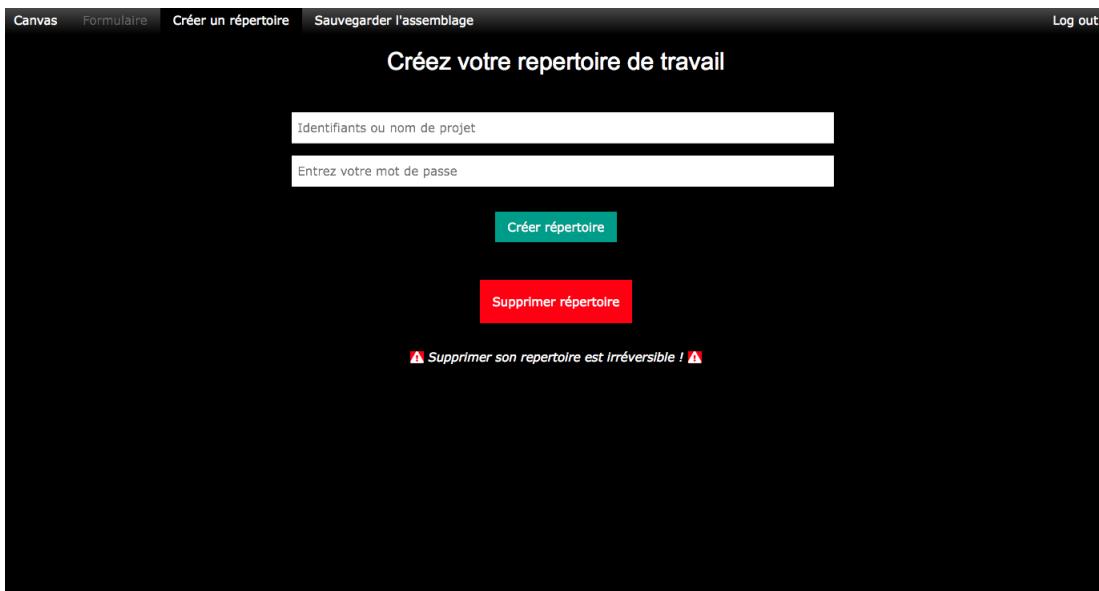


FIGURE 3.7 – Création d'un répertoire de travail

3.7.2 Suppression d'un répertoire de travail

Dans le même onglet, l'utilisateur peut supprimer son répertoire à condition qu'il soit bien connecté à celui-ci quand il lance la requête. L'envoi d'une requête http est déclenché vers l'url '/secure/rd'. Le serveur vérifie que l'utilisateur n'est pas 'main'. En effet, cet utilisateur est réservé à la création de répertoire. La fonction va alors chercher le nom de l'utilisateur dans la table des utilisateurs (créé sur le serveur à partir du fichier passwd.json) et va le supprimer (même opération pour le mot de passe).

Le nom de l'utilisateur et les références associées à ce nom sont ensuite supprimés de la table des projets (créée à partir de projects.json). Les deux fichiers d'où sont tirés ces structures de données sont ensuite sauvegardés et l'utilisateur est redirigé vers la page d'authentification. Si l'utilisateur n'est pas connecté ou s'il est connecté au répertoire 'main', alors il est redirigé vers l'interface.

3.7.3 Enregistrement d'un assemblage

Dans l'onglet Exporter un assemblage, géré par le contrôleur "UploadImage", l'utilisateur a la possibilité de générer un aperçu de l'assemblage qu'il souhaite créer, sous forme d'une miniature. Pour cela, la fonction **genThbCanvas()**, reliée au bouton "Aperçu" par la directive **ng-click**, utilise la méthode **HTMLCanvasElement.toDataURL()** qui renvoie une data : URL contenant une représentation de l'image au format spécifié par l'argument type (ici : image/png). L'image renvoyée est en 96dpi. La fonction utilise ce retour de cette méthode pour définir la source de l'aperçu en utilisant l'attribut id="thumbnailCanvas" pour sélectionner l'image donc l'emplacement qui accueillera l'aperçu généré. L'aperçu est miniaturisé grâce au fichier sidebar.css qui réduit la taille de tous les éléments images de l'interface.

Il est également possible d'uploader la nouvelle image sur le serveur via la fonction **UploadCanvas**, du même contrôleur, qui est déclenchée via la directive **ng-click**. Elle envoie une requête http (méthode POST) vers le serveur à l'url "/secure/compUpload".

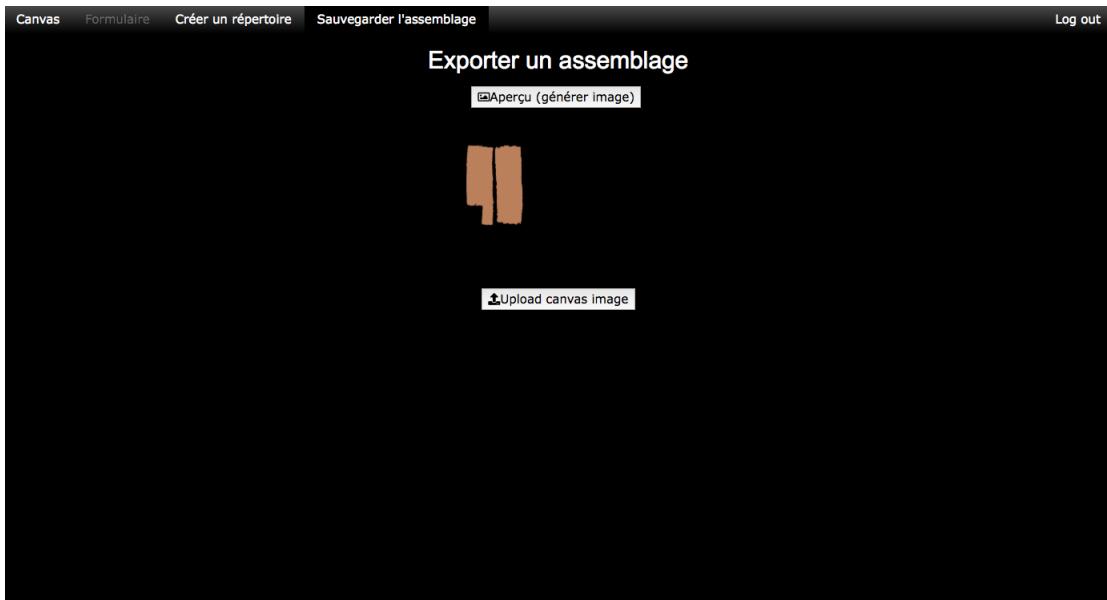


FIGURE 3.8 – Enregistrement d'un assemblage de fragments

À partir de là, le serveur vérifie que l'utilisateur est bien authentifié, qu'il ne soit pas identifié sous le répertoire main, et qu'il ait bien envoyé des paramètres à traiter par le serveur. L'image est alors récupérée en Base64, la source est également récupérée.

Ensuite, l'image est ré-écrite en binaire à partir de la Base64 et enregistrée dans le dossier contenant toutes les autres images. L'image va ensuite être rognée pour ne garder que les images formant l'assemblage et pas le reste du canvas. Cette action est performée par l'outil *Imagemagick*, un outil de manipulation d'image en ligne de commande. Son utilisation est rendue possible grâce à la fonction `exec()` qui permet d'exécuter une ligne de commande.

Les références et propriétés des images faisant parties de l'assemblage, sont sauvegarder dans un fichier JSON du même nom que le fichier image associé.

La table de projets est ensuite mise à jour et le fichier d'origine (`projects.json`) est également modifié. La table contenant toutes les références et les sources utilisables est également mise à jour et le fichier `PapyrusTable.json` est modifié en conséquences.

3.7.4 Suppression d'images du serveur/répertoire

Le menu déroulant, qui apparaît lorsque l'utilisateur clique sur la <div> correspondant à un fragment, propose de supprimer un fragment du répertoire de travail. Un clic gauche sur cette option va déclencher la fonction `RemoveImageServer(item.Ref)` qui va effectuer l'envoi d'une requête https de type POST vers le serveur à l'url "/secure/removeImg". Le serveur va alors vérifier que l'utilisateur est bien authentifié qu'il n'est pas identifié en tant que 'main', qui est réservé à la création, et s'assure que l'utilisateur envoie bien des paramètres.

Il vérifie ensuite que la référence est bien présente dans la table contenant toutes les références et les sources. Si cette condition est satisfaite et que le propriétaire de ce fragment correspond à l'utilisateur, alors est stocké dans un tableau qui sera utilisé pour définir les chemins des fichiers image associés à ce fragment et qui sont supprimés du répertoire Datas. La table des références est mise à jour en supprimant l'objet en question et le fichier `projects` est également mis à jour.

3.8 Mise en place d'éléments de sécurité

Une politique de sécurisation du serveur est indispensable à toute application exposée à des cyber-menaces. Le serveur et le client possèdent plusieurs protections contre le sniffing et le spoofing.

3.8.1 Utilisation exclusive du protocole HTTPS

Le serveur est configuré pour fonctionner exclusivement en HTTPS. Le port par défaut est 8443. Il s'agit d'un port ne nécessitant pas les priviléges administrateurs.

3.8.2 Certificats de sécurité

Les certificats de sécurité permettent de créer une connexion privée et sécurisée entre le client et le serveur. L'algorithme de chiffrement des données utilisé est *Elliptic curve digital signature algorithm (ECDSA)*. La courbe utilisée dans l'algorithme ECDSA est secp256k1. Les certificats sont générés par l'OS du serveur avant le lancement de celui-ci.

3.8.3 Sécurisation des en-têtes HTTP

La sécurisation des en-tête HTTP est une des premières protections contre les menaces. Ces mécanismes permettent au client et au serveur de fonctionner ensemble en confiance. L'en-tête indique au client d'utiliser la protection XSS du navigateur, met en place la protection contre les attaques de types CSRF, interdit les liaisons en HTTP sans SSL par exemple.

Voici les paramètres de requête fixés par défaut dans le serveur :

- Cache-Control : no-store, no-cache, must-revalidate, proxy-revalidate
- Pragma : no-cache
- Expires : 0
- Surrogate-Control : no-store
- X-DNS-Prefetch-Control : off
- X-Frame-Options : SAMEORIGIN
- X-Download-Options : noopen
- X-Content-Type-Options : nosniff
- X-XSS-Protection : 1 ; mode=block
- X-CSRF-Token : \$argon2i\$v=19\$m=4096,t=3,p=1\$<HASH>
- Strict-Transport-Security : max-age=15552000 ; includeSubDomains
- Accept-Ranges : bytes
- ETag : W/"XXXXXXXX"
- Feature-Policy : camera : 'none' ; payment : 'none' ; microphone : 'none'
- Public-Key-Pins : 'pin-sha256=<KEY1BASE64>' ; pin-sha256=<KEY2BASE64> ; pin-sha256=<KEY3BASE64>' ; max-age=36500'

3.8.4 Protection contre les attaques CSRF

La protection contre les attaques CSRF est construite de la manière suivante : le serveur et le client partagent une clé secrète présente dans chaque échange risqué. Ici la phrase secrète

est intégrée aux en-têtes HTTP dans le paramètre X-CSRF-Token. Le serveur enregistre la clé dans le fichier de session de l'utilisateur. Il compare ensuite la clé enregistrée à la clé fournie par le client dans ses requêtes. Les requêtes sensibles protégées par ce procédé sont toutes les requêtes POST sauf celles du login et du prototype pour les métadonnées.

3.8.5 Protection des mots de passe

Les mots de passe sont hashés dans le serveur à l'aide de l'algorithme Argon2i. Cet algorithme est la nouvelle référence des fonctions de hashage. Le serveur crée le hash du mot de passe à la création d'un répertoire et vérifie la concordance des mots de passe quand l'utilisateur tente de s'identifier.

3.9 Manuel utilisateur

Afin de permettre aux utilisateurs de se familiariser rapidement avec l'interface, un manuel utilisateur a été créé. Ce manuel utilisateur s'adresse autant aux archéologues qu'aux développeurs souhaitant continuer ou améliorer le développement de l'interface.

A l'intention des utilisateurs de l'interface le manuel détaille les différents éléments de l'interface ainsi que les fonctionnalités proposées par l'interface.

Une seconde partie, destinée aux développeurs, présente l'ensemble des systèmes de sécurité rattachés à l'interface. Il apporte également des précisions sur le retour des tests de sécurité effectués. Un modèle de bonnes pratiques pour l'intégration de nouveaux scripts côté serveur y est détaillé. Enfin, un protocole d'ajouts de fonctionnalités dans les composants Aréa côté client est expliqué.

3.10 Perspectives

3.10.1 Modification des métadonnées

Un onglet de modification des métadonnées a été implémenté côté client. En effet l'onglet prend la forme d'un formulaire HTML qui demande à l'utilisateur de rentrer des informations tel que le type d'image, ses dimensions, son origine et ajouter des commentaires. Un prototype de fonction pour la gestion de ce formulaire a bien été implémenté côté serveur afin de traiter la requête.

Néanmoins de part la complexité de l'organisation des données XML et de l'aspect non obligatoire pour la réalisation d'une application fonctionnelle et répondant aux besoins du client, la fonctionnalité est à ce jour non opérationnelle.

3.10.2 Options pratiques et d'ergonomie

-Langues : Anglais Allemand

Le changement de langues pourrait être implémenté grâce à l'utilisation de variable de texte en début de fichiers.

-Ergonomie pour les utilisateurs gauchers

Afin de faciliter encore plus la manipulation de l'interface par l'utilisateur, une ergonomie inversée pourrait être implémenté pour les gauchers. Grâce à un bouton l'utilisateur pourrait alors choisir d'obtenir l'interface en "miroir", la colonne contenant les miniatures se trouverait alors sur la droite.

3.10.3 Choix des images pour l'implémentation

À ce jour, l'ensemble de la base de données fournie est implémentée dans le serveur de l'interface, et toutes les images sont accessibles directement via la colonne de gauche. Ceci fonctionne grâce à la taille minime des données à disposition. Par la suite, une amélioration de l'interface pourrait être réalisé en proposant à l'utilisateur de choisir un jeu de données sur lequel il souhaite travailler. Il s'agira alors "d'alléger" les choix possibles dans la colonne de gauche et de permettre à l'utilisateur de n'utiliser que les données de son intérêt. Le serveur embarque déjà ces fonctionnalités mais l'interface ne propose pas cette option dans la version 1.0.

Conclusion

Il s'agissait ici de développer une interface web pratique et fonctionnelle afin de faciliter le travail de reconstruction des Papyrus par les archéologues. Ceci permet à l'utilisateur de mieux visualiser les assemblages possibles entre les fragments. L'interface permet d'effectuer diverses manipulations en facilitant l'accès aux informations liées aux fragments manipulés. L'interface permet l'exécution de divers scripts qui les aideront dans la reconstitution. Toutes ces fonctionnalités sont implémentées afin que les archéologues puissent réaliser et sauvegarder les assemblages de leur choix.

Suite à une analyse des différents enjeux de ce projet et des outils déjà existants à disposition, il a pu être mené à bien. Il prend la forme d'une interface web développée grâce au trio de langages de programmation web HTML5 / CSS3 / JavaScript côté client, avec l'aide du framework AngularJS. Ce framework permet de structurer plus clairement l'application et ainsi de faciliter le développement.

De plus, NodeJS est utilisé en tant que serveur. Cette interface permet donc aux archéologues, à partir d'un jeu de fichiers images et xml, de manipuler à leur guise des fragments de papyrus, indépendamment ou conjointement, en créant des assemblages. Les algorithmes intégrés à l'application sont : "threshold" pour le traitement d'images, et "score" calculant le score de probabilité d'assemblage et permettant leurs correctes associations. Le serveur est modulaire et offre des possibilités d'extensions.

Bibliographie

- [1] <https://gmic.eu/reference.shtml>.
- [2] <https://www.archeosites.fr/carte-generale/c/0>.
- [3] <https://www.tiobe.com/tiobe-index/>.
- [4] <https://pypl.github.io/PYPL.html>.
- [5] <https://angularjs.org/>.
- [6] <https://www.w3.org/html/>.
- [7] <https://nodejs.org>.
- [8] <https://www.metasploit.com/>.
- [9] <https://www.exploit-db.com/>.
- [10] https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project.
- [11] <https://portswigger.net/burp/>.
- [12] <https://www.root-me.org/?lang=en>.
- [13] <https://www.hackthebox.eu/>.