

UNIVERSITÉ DE BORDEAUX - MASTER 1 BIOINFORMATIQUE

# ARCHEO ASSEMBLER.

## Projet de Programmation

---

Développement d'une interface Web pour la visualisation  
d'artefacts et la simulation d'assemblage de fragments

Jlajla Hamza, Pétrus Louis, Jelin Rémy, Luxey Victor, Pratx Julie,  
Lesourd-Aubert Valentine

---

Responsable de projet : Mme Cécilia Ostertag

29 mai 2020

# Résumé

Les réseaux de neurones artificiels sont de plus en plus utilisés dans de nombreux domaines scientifiques. Ils permettent de remplacer certains travaux encore laborieux tels que la reconstruction automatique de puzzles ou de documents par des modèles algorithmiques. Ces algorithmes sont en effet capables d'extraire des représentations graphiques à partir d'images et peuvent ensuite être utilisés pour distinguer les fragments pouvant s'associer.

L'archéologie est un des domaines scientifiques qui peut bénéficier de cette nouvelle approche. Habituellement les archéologues extraient, lors de fouilles, des fragments de fabrications antiques, qu'ils doivent ensuite reconstituer à la main. Un algorithme de ré-assemblage automatique utilisant cette nouvelle technologie est alors parfaitement adapté.

# Table des matières

<b>Liste des tableaux</b>	<b>ii</b>
<b>Table des figures</b>	<b>iii</b>
<b>Liste des abréviations</b>	<b>iv</b>
<b>Introduction</b>	<b>1</b>
<b>1 Analyse</b>	<b>2</b>
1.1 Etat de l'art . . . . .	2
1.2 Présentation du sujet . . . . .	3
1.3 Analyse des besoins . . . . .	4
1.3.1 Besoins fonctionnels . . . . .	4
1.3.2 Besoins non-fonctionnels . . . . .	4
<b>2 Conception de l'interface</b>	<b>6</b>
2.1 Architecture globale . . . . .	6
2.2 Récupération et conversion des données . . . . .	7
2.3 Affichage des données et des images d'artefact . . . . .	7
2.4 Filtrage des données . . . . .	7
2.5 Sélection d'artefact . . . . .	8
2.6 Édition des données . . . . .	8
2.7 Intégration des algorithmes Python . . . . .	8
2.7.1 Transpileur . . . . .	8
2.7.2 Interpréteur Python . . . . .	8
2.7.3 WebAssembly . . . . .	8
2.7.4 WSGI . . . . .	9
<b>3 Réalisation</b>	<b>10</b>
3.1 Récupération et conversion des données . . . . .	10
3.2 Affichage des données et des images d'artefact . . . . .	11
3.3 Filtrage des données . . . . .	11
3.3.1 Choix possibles . . . . .	11
3.3.2 Actualisation des données . . . . .	11
3.4 Sélection d'artefact . . . . .	11
3.5 Édition des données . . . . .	12
3.5.1 Ajout . . . . .	12

3.5.2	Modification . . . . .	12
3.5.3	Suppression . . . . .	12
3.6	Intégration du code Python . . . . .	12
3.6.1	Environnement virtuel et modules . . . . .	13
3.6.2	Architecture globale . . . . .	13
3.6.3	Script jstop.py et son fonctionnement . . . . .	14
3.6.4	La problématique du binding Python/JavaScript . . . . .	14
<b>4</b>	<b>Installation et tutoriel d'utilisation</b>	<b>15</b>
4.1	Pré-requis . . . . .	15
4.2	Créer son environnement virtuel en python2 . . . . .	15
4.3	Installation des fichiers . . . . .	15
4.3.1	Chargement des méta-données . . . . .	16
4.3.2	Interactions de l'utilisateur avec l'interface . . . . .	17
4.3.3	Résultats . . . . .	17
	<b>Conclusion générale</b>	<b>20</b>

# Liste des tableaux

# Table des figures

1.1	Exemple d'ostracon inscrit de textes hiératiques . . . . .	3
2.1	Architecture des fichiers du projet . . . . .	6
3.1	Architecture globale du projet avec le module Flask . . . . .	14
4.1	Lancement du serveur flask . . . . .	16
4.2	Interface client en se rendant sur l'url du serveur local flask . . . . .	16
4.3	Chargement des données avec l'interface web . . . . .	17
4.4	Sélection des ostraca d'intérêts . . . . .	17
4.5	Résultats des ostraca d'intérêts . . . . .	18

# Liste des Abréviations

**CSS** : Cascading Style Sheets

**CSV** : Comma-Separated Values

**DOM** : Document Object Model

**ES** : ECMAScript

**HTML** : Hypertext Markup Language

**JSON** : JavaScript Object Notation

**NPZ** : NumPy Zipped

**OCHRE** : Online Cultural and Historical Research Environment

**ODS** : Open Document Spreadsheet

**SNN** : Siamese Neural Network

**TEI** : Text Encoding Initiative

**WSGI** : Web Server Gateway Interface

**W3C** : World Wide Web Consortium

**XHTML** : Extensible HyperText Markup Language

**XLS** : eXcel Spreadsheet

**XML** : eXtensible Markup Language

**XSLT** : eXtensible Stylesheet Language Transformations

**XSD** : XML Schema Definition

# Introduction

Dans le cadre de notre première année en Master de Bio-informatique à l'Université de Bordeaux, nous avons réalisé un projet de programmation nous permettant de mettre en pratique nos connaissances et nos compétences acquises.

Les réseaux de neurones artificiels sont de plus en plus utilisés pour des problèmes de reconstruction automatique de puzzles ou de documents. L'interface Web développée pour ce projet vise à faciliter l'utilisation d'un algorithme de ré-assemblage d'artefacts. Cet algorithme, basé sur les graphes et les réseaux de neurones, propose aux scientifiques des reconstructions 2D de poteries à partir de fragments récupérés lors de fouilles archéologiques.

Ce rapport est composé de 3 chapitres : l'**analyse**, la **conception de l'interface** et sa **réalisation**.

L'**analyse** présentera brièvement le sujet du projet et définira les différents besoins fonctionnels et non fonctionnels constituant nos objectifs de conception et réalisation de l'interface Web.

Dans le chapitre **conception de l'interface** sera fourni une description de l'architecture du projet ainsi que l'organisation des principaux axes de réflexion autour des problématiques suivantes rencontrées : le chargement des données, le filtrage et l'édition des données, la sélection des artefacts et l'intégration des algorithmes Python.

Enfin, le développement final de l'interface ainsi que son fonctionnement seront détaillés dans le chapitre **réalisation de l'interface**.

Les difficultés que nous avons rencontrées au cours du développement du projet, ont nécessité un approfondissement de nos connaissances du web, notamment l'exploration de la programmation côté serveur. Cela a aboutit à une réorganisation de notre travail autour des deux aspects du web que sont le côté client et le côté serveur. L'état actuel du projet est encore en phase de développement, et il a été décidé de proposer deux versions avec et sans intégration des algorithmes Python du client.

**Mots clés** : Archéologie, Ostracon, Réseaux Neuronaux Siamois, Ré-assemblage, Interface Web, Graphes

**Accès au code du projet** : <https://github.com/lpetrus32/ArcheoAssembler.git>

**Accès au code du projet compatible avec le module Python Flask** : [https://github.com/RJ0wly/ArcheoAssembler\\_Flask](https://github.com/RJ0wly/ArcheoAssembler_Flask)

# Chapitre 1

# Analyse

## 1.1 Etat de l'art

Les débuts de l'utilisation des outils informatiques dans la recherche archéologique, dans les années 80, étaient focalisés sur des problématiques d'inventaire. Les bases de données jouaient alors un rôle prépondérant. Encore aujourd'hui, les bases de données constituent un moyen extrêmement fiable de stocker durablement les données d'excavation et permettent, en autre, de contrôler l'accès aux ressources pour des travaux de recherches.

Le progrès de l'informatique et la multiplication de projets visant à répondre à des problématiques variées dans ce domaine sont à l'origine de l'émergence de plateforme d'outils computationnels comme OCHRE [2], capable de travailler avec des données récemment acquises jusqu'à l'archivage de celles-ci.

Avec le développement de l'intelligence artificiel, il est possible de reconstruire des images à partir de fragments grâce à des algorithmes de reconstruction à apprentissage automatisé qui utilisent la logique et les mathématiques cachés derrière les puzzles, le Machine Learning. Cette méthode se base sur des approches statistiques, pour donner à un programme la capacité d'"apprendre" à l'aide de données d'entraînement, afin d'améliorer sa précision.

Les Réseaux de Neurones Siamois, de l'anglais Siamese Neural Networks (SNN) sont un type de réseaux de neurones artificiels utilisés en Machine Learning pour résoudre des problèmes de reconnaissances d'images (Bromley et al., 1993)[5]. Il existe de nombreuses bibliothèques open source permettant de développer et d'exécuter des applications de Machine Learning comme des réseaux neuronaux : *OpenNeuralNetwork* pour le langage C++, en langage python *TensorFlow* et *Scikit-Neural Network* ou encore *Brain.js* en langage Javascript.

L'algorithme Python proposé par C.Ostertag est capable de proposer une reconstruction de poteries en 2D (C.Ostertag et al., 2020) [6] grâce à un SNN.

## 1.2 Présentation du sujet

Le but de l'algorithme développé par C.Ostertag est d'apporter aux archéologues une alternative automatisée à la reconstruction de poteries antiques, à partir de fragments - communément appelés "artefacts" - récupérés lors de fouilles archéologiques. Dans le cadre du projet, nous travaillons sur des tessons de poteries appelés "ostraca", utilisés comme support d'écriture par les scribes de l'Egypte antique. Le modèle sur lequel est basé l'algorithme a été pré-entraîné à reconnaître et exploiter les différentes inscriptions existantes au sein d'un même ostracon, ainsi que les différences de texture entre ostraca.

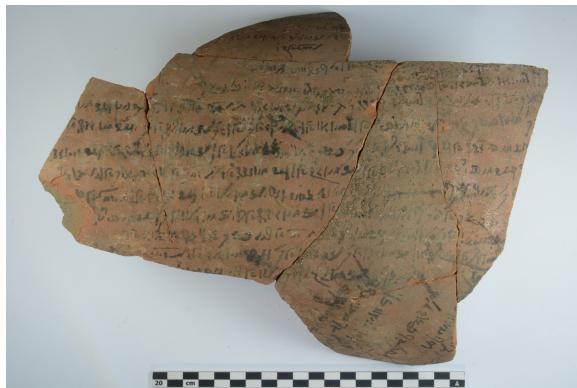


FIGURE 1.1 – Exemple d'ostracon inscrit de textes hiératiques

L'objectif principal de ce projet de programmation est la réalisation d'une interface Web, à partir de laquelle il sera possible d'interagir avec des données de fouilles, des informations textuelles ou "méta-données", et des images ainsi que d'établir le lien avec les algorithmes, écrit en langage Python.

L'ergonomie de l'affichage doit permettre à l'utilisateur de retrouver et sélectionner des fragments d'intérêt, en fonction des différents paramètres disponibles, pour créer un jeu de données spécifiques à l'analyse souhaitée.

Deux utilisations des algorithmes sont envisageables, classiquement le ré-assemblage des images préalablement sélectionnées par l'utilisateur. Mais aussi la création de patches, un ensemble d'images secondaires, résultants de la fragmentation d'une seule image. Ces patches constituent les exemples utilisés pour remplir le SNN. La sortie de l'algorithme renvoie à l'interface des propositions d'assemblage des patches, en tant que résultats des prédictions du modèle.

L'interface doit garantir une utilisation optimale de l'algorithme ainsi qu'une lecture claire des résultats obtenu, sous forme de graphes.

## 1.3 Analyse des besoins

### 1.3.1 Besoins fonctionnels

Les fonctionnalités essentielles sont :

- Ajout, suppression et édition des métadonnées,
- Choix manuel des filtres à appliquer sur le tableau de métadonnées parmi les paramètres disponibles dans le-dit tableau afin de trier les fragments,
- Passage possible entre la vue du tableau de métadonnées et la vue des images de fragments correspondants pour faciliter leur sélection,
- Lancement de l'algorithme de création d'un jeu de données artificiel à partir de patches extraits des images du tableau,
- Visualisation des graphiques/résultats et interaction avec eux.

### 1.3.2 Besoins non-fonctionnels

En ce qui concerne la présentation de l'interface Web, nous utilisons les langages **HTML** et **CSS**. Pour un rendu interactif et dynamique, des fonctionnalités sont implémentées (boutons, menus déroulants, etc.) en langage **Javascript**. Les images d'ostraca en lien avec les métadonnées doivent respecter le format **NPZ**. La page Web est compatible avec différents navigateurs (Google Chrome, Mozilla et Safari).

#### Langages

##### HTML :

- HTML signifie Hyper Text Markup Language.
- Permet la création de pages Web en respect des standards développés par le W3C.
- HTML se compose d'éléments ou "tags" qui stockent un contenu divers.
- A un tag, peuvent être spécifiés des attributs qui permettent de contrôler le rendu de son contenu.
- Le navigateur reconnaît les tags et affiche uniquement leur contenu, avec le rendu adéquate.
- Au moment du chargement de la page Web, le navigateur génère le DOM (Document Object Model) HTML.
- Le DOM est un standard du W3C et une interface de programmation permettant d'interagir avec les tags.

##### CSS :

- CSS pour Cascading Style Sheet ou feuilles de style en cascade, est un langage de feuille de style.
- Il est utilisé pour décrire la présentation d'un document écrit en HTML ou en XML.
- Il décrit la façon dont les éléments doivent être affichés à l'écran (polices, couleurs, espacements, etc.).
- C'est l'un des langages principaux du Web ouvert.

##### JavaScript :

- Javascript, nommé alternativement ECMAScript, est le langage de programmation du Web.
- Langage de script, léger et orienté-objet, utilisant le concept d'héritage de classes ou "prototypage".
- Sa version actuelle (ES6) est supportée par la plupart des navigateurs modernes.
- Tous les navigateurs intègrent nativement un interpréteur javascript.

- Permet la création de pages Web dynamiques, en suivant le paradigme de programmation événementielle.
- Possède des outils lui permettant d’interagir avec des balises HTML/XML par l’intermédiaire du DOM.

#### **XML :**

- XML signifie eXtensible Markup Language.
- Conçu pour le stockage et le transport de données non-sécurisées.
- Reconnu officiellement par le W3C et intégré au projet HTML5.
- Comme HTML il appartient à la famille des langages à balises et possède son propre DOM.
- A l’instar de CSS, il permet une séparation de l’information des données et de leur affichage.
- Léger, portable et utilisable aussi bien côté client que côté serveur.
- Syntaxe aisément compréhensible tant par l’homme que par la machine.
- Bonne alternative aux systèmes de gestion de base de données tant que les données ne sont pas sensibles.
- Peut être associé à des schémas XSD pour définir les contraintes imposées aux données.

#### **Formats**

##### **NPZ :**

- Format d’archive zip traité avec la bibliothèque NumPy [9],
- Stockage des données d’une liste à l’aide de la compression gzip,
- un objet NpzFile est renvoyé, de type semblable aux dictionnaires.

#### **Bibliothèques**

##### **SheetJS js-xlsx :**

- Analyseur syntaxique implémentant la lecture, l’écriture et l’export de données tabulaires.
- Supporte différents formats de feuille de calcul, dont .xls, .ods et .csv.
- Compatible avec les navigateurs supportant ES3+ (ie : tous les navigateurs actuels).
- Existe en version professionnelle ou communautaire (libre)

##### **Modules Python :**

- NetworkX permet la création et la manipulation de graphe.
- Tensorflow et Keras permettent la création/utilisation de modèles de machine learning et de deep learning.
- NumPy et Pandas permettent la création et la manipulation de structures de données sur lesquelles il est possible d’appliquer des fonctions mathématiques avancées.
- OpenCV permet l’importation ou l’écriture d’image sous la forme npz dans le script Python.
- SciPy vise à unifier et fédérer un ensemble de bibliothèques Python à usage scientifique, en utilisant les tableaux et matrices du module NumPy.
- Matplotlib est destiné à tracer et visualiser des données sous formes de graphiques.
- Flask est un micro-framework open-source de développement web, permettant d’intégrer des scripts Python côté serveur et de communiquer avec une page HTML/JavaScript.

# Chapitre 2

## Conception de l'interface

L'interface du projet **ArcheoAssembler** est conçue dans le but de faciliter l'application des algorithmes de création et d'assemblage de patches à une sélection de données. Afin de répondre aux besoins évoqués lors de l'analyse du projet, nous utiliserons les langages Javascript, CSS et HTML.

### 2.1 Architecture globale

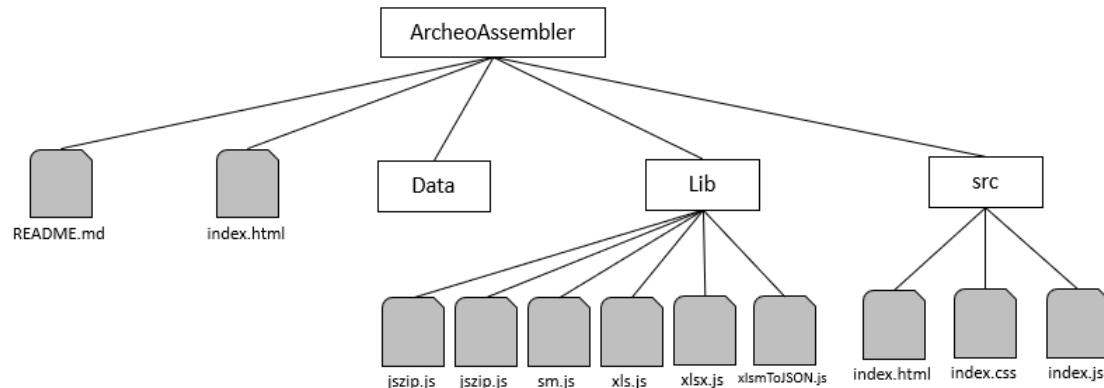


FIGURE 2.1 – Architecture des fichiers du projet

## 2.2 Récupération et conversion des données

La première étape consiste à importer un fichier de données que nous devrons convertir au format XML pour en extraire et transformer les valeurs et les attributs dans une forme plus facilement exploitable.

L'utilisateur pourra charger son fichier sur l'interface en respectant les formats suivants pris en charge : .XML, .XLS, .XLSX, .ODS et .CSV.

En ce qui concerne les formats .XLS, .XLSX, .ODS et .CSV, une conversion au format XML sera nécessaire. Le format XML étant utilisé comme outil de référence dans le partage standardisé des données en archéologie (initiative TEI [1], 2002).

Nous souhaiterions prendre en compte les données ainsi récupérées dans des objets Javascript regroupant les valeurs des attributs pour chaque ligne du tableau. Nous pourrions également récupérer les attributs et les valeurs du fichier XML pour les stocker dans des listes.

Le fichier ou la variable .XML devra être conservé en vue d'une modification possible.

La récupération des images se fera de la même manière que la récupération des données, via un bouton d'importation de fichier.

## 2.3 Affichage des données et des images d'artefact

Une fois le fichier de données chargé, nous afficherons les données dans un tableau. Ainsi, nous pourrons individualiser chaque ligne par la création d'un identifiant.

L'affichage des images se basera sur le travail réalisé précédemment pour l'affichage des données. Un tableau devrait permettre d'afficher une à une les images représentant les ostraca.

## 2.4 Filtrage des données

### Choix possibles

Les fichiers de métadonnées sont souvent volumineux, à l'image de la quantité d'artefacts retrouvés par les archéologues et des informations qui y sont liées.

L'utilisateur pourra, parmi les attributs possibles, choisir quels sont ceux qui vont lui permettre d'accéder facilement aux artefacts qu'ils souhaitent sélectionner. Il pourra compléter son filtrage selon la valeur qu'il recherche pour cet attribut en particulier.

Plusieurs filtres pourront être ajoutés. Une fonction de suppression des filtres appliqués sera également implémentée.

### Actualisation des données

Le tableau de données ainsi que les images doivent constamment être actualisés en fonction des actions de filtrage de l'utilisateur. S'il choisit d'ajouter ou bien de supprimer un filtre, les valeurs devant être affichées sur l'interface ne seront pas les mêmes.

## 2.5 Sélection d'artefact

L'utilisateur aura la possibilité de cliquer sur les lignes du tableau de métadonnées ou les images pour enregistrer le fragment sélectionné.

Les images de fragments sélectionnés seront affichées dans un bloc particulier de l'interface pour que l'utilisateur puisse à chaque instant visualiser sa sélection.

## 2.6 Édition des données

Pour permettre à l'utilisateur d'édition ses données via l'interface nous implémenterons un mode "édition" qui permettra, lorsqu'il sera activé, d'ajouter un ou plusieurs artefacts et de sélectionner une ligne du tableau de données pour la modifier ou la supprimer.

## 2.7 Intégration des algorithmes Python

Notre code, qui gère les interactions avec l'interface côté client, étant écrit en langage JavaScript, la difficulté est de trouver un moyen de communication entre ces deux langages.

### 2.7.1 Transpileur

Notre idée de départ est d'utiliser un transpileur, c'est-à-dire, un type de compilateur qui prend le code source d'un langage de programmation et le compile dans un autre langage de programmation. Dans notre cas il s'agirait donc de "convertir" python en JavaScript. Les transpileurs Python-JavaScript les plus utilisés actuellement sont Transcrypt et Brython.

### 2.7.2 Interpréteur Python

Une seconde idée peut être d'utiliser un interpréteur python, c'est un type de compilateur qui, comme un transpileur, va interpréter le langage Python en langage JavaScript. Cependant, à la différence d'un transpileur, pour fournir une véritable expérience Python, un interpréteur Python va fournir un environnement d'exécution dans lequel le code compilé s'exécute et s'affiche sur la page web en temps réel. Un des problèmes majeur de cette idée est que l'environnement ne pourra pas exécuter des modules externes à Python, tel que Tensorflow, et sera limité à faire tourner des fonctions rudimentaire de Python. Un interpréteur sert à exécuter du code Python en direct sur une page Web, sans nécessiter l'installation de ce langage, ainsi que "faire tourner" des courts programmes. Les interpréteurs Python les plus connus sont, actuellement, PyPy.js et Pyodide.

### 2.7.3 WebAssembly

Une troisième possibilité est un nouveau type de code : WebAssembly. C'est un langage bas niveau, semblable à l'assembleur permettant d'atteindre des performances proches des applications natives (par exemple écrites en C/C++) tout en fonctionnant sur le Web. WebAssembly est conçu pour fonctionner en lien avec JavaScript. L'avantage de ce nouveau code est qu'il permet l'exécution d'un code écrit dans un autre langage que JavaScript dans une page web. Les modules WebAssembly peuvent être importés en exposant

les fonctions WebAssembly à utiliser via JavaScript. Cependant pour le moment les modules WebAssembly ne permettent pas d'intégrer du code Python.

#### 2.7.4 WSGI

Notre dernière solution est d'utiliser un module compatible WSGI comme Flask, c'est un module Python qui permet la communication entre une application web et un serveur. Un des intérêts d'un WSGI c'est de permettre l'utilisation de script utilisant des modules externes tel que Keras, Numpy, NetworkX et de communiquer des données JavaScript interprétable en Python grâce au format JSON pour ensuite permettre afficher les résultats des scripts sur une page web.

Au vu des différentes possibilités actuelles permettant l'intégration de scripts Python dans une interface web écrit en langage JavaScript, la solution qui semble la plus adapté est l'utilisation d'un WSGI, ici Flask, qui permettra la communication entre l'interface web et les scripts Python. De plus les scripts de machine learning du client utilisant de nombreux modules externes sont compatibles avec le module Flask.

# Chapitre 3

## Réalisation

### 3.1 Récupération et conversion des données

Un bouton implémenté en HTML permet à l'utilisateur de charger son fichier de données. Lorsque le programme gère un fichier en entrée comportant l'extension .XLS, .CSV ou .ODS, la conversion nécessaire en XML est effectuée à l'aide d'une méthode du type "XLS to XML".

Pour ce faire, un fichier d'extension .XLS ou .ODS doit d'abord passer par une étape de conversion en fichier d'extension .CSV. Une fonction d'assistance fournie dans XLSX.utils de la bibliothèque **SheetJS js-xlsx** permet de lire le fichier et la méthode XLS.utils.make\_csv() génère une feuille de calcul au format CSV.

Le "Document Object Model" (DOM), plus particulièrement le DOM XML, définit une manière standard d'accéder et de manipuler les documents XML. Il présente un document XML sous forme d'arborescence. Par l'analyse d'un document XML avec un analyseur DOM, nous obtenons un accès à cette arborescence qui contient tous les éléments du document. Avec l'analyseur DOMParser, le programme récupère les attributs et les valeurs de chaque ligne correspondante à un artefact et stocke ces informations dans des listes.

En définitif, les modifications ultérieures des données n'auront pas lieu sur le fichier ou la variable XML mais sur des listes secondaires créées à partir des listes initiales comportant les données chargées en début de projet par l'utilisateur.

Préalablement au chargement du fichier d'images, l'utilisateur doit placer son fichier dans le sous-dossier "Data" du dossier contenant l'ensemble des données utiles au projet. En effet, le programme n'est en mesure de récupérer les images que si elles sont placées dans un des fichiers du projet. Cette étape est nécessaire puisque nous utilisons le chemin relatif du dossier d'images pour les récupérer. Par ce moyen, aucun problème de sécurité concernant l'utilisation des données provenant de l'ordinateur de l'utilisateur n'est rencontré.

La récupération des métadonnées ne pose pas cette contrainte puisqu'après les avoir récupérés, ces données sont sauvegardées dans des listes pour une éventuelle réutilisation de ces données.

En chargeant son fichier d'images par le bouton Parcourir, nous pouvons, à partir du nom du fichier, écrire le chemin relatif qui nous donnera accès au fichier. Les noms des images sont parcourus et sauvegardés dans une liste.

## 3.2 Affichage des données et des images d'artefact

Les attributs et les valeurs des artefacts sont affichés sur l'interface dans un tableau. Les attributs et les valeurs utilisées sont celles du fichier XML qui ont été stockées dans des listes. Lorsque l'utilisateur déplace le curseur sur une ligne, le type de curseur change comme la couleur de la ligne, ce qui indique qu'au clic de l'utilisateur, l'ostraca correspondant est enregistré avec les autres artefacts sélectionnés. Chaque ligne est liée par le biais d'un identifiant à la fonction **Selection()**.

Contrairement aux données, les images ne sont finalement pas affichées dans un tableau mais elles sont ajoutées les unes à la suite des autres.

## 3.3 Filtrage des données

### 3.3.1 Choix possibles

Sachant qu'un fichier de méta-données peut comprendre de nombreux attributs, nous proposons à l'utilisateur de choisir un attribut sur lequel il veut appliquer un filtre, et ainsi lui permettre d'affiner sa recherche en affichant uniquement les artefacts dont les valeurs respectent le filtre appliqué. Pour cela, nous utilisons la liste d'attributs enregistrés au chargement des données afin de mettre en place une liste-déroulante dans le menu de filtrage.

L'utilisateur doit alors définir une valeur pour l'attribut choisi. Si aucune valeur n'est entrée, un message d'erreur s'affichera à l'écran.

Concernant la valeur, deux possibilités ont été envisagées. Dans le cas où les valeurs sont numériques, nous avons implémenté un champ de saisie permettant d'entrer des opérateurs logiques. Seules les lignes qui respectent la condition seront visibles. Une seconde option consiste à donner la possibilité à l'utilisateur de choisir une valeur particulière. Une deuxième liste déroulante contenant les valeurs de l'attribut est nécessaire. Cette liste est dynamique, c'est-à-dire actualisée en fonction de l'attribut choisi en première étape. Le programme parcourt la liste des ostraca affichés à l'écran et, si la condition est respectée pour une ligne, cette dernière est ajoutée à une liste temporaire. Une fois toutes les lignes parcourues, la liste temporaire devient la nouvelle liste d'ostraca à afficher.

Les filtres appliqués sont affichés sur l'interface, et peuvent au choix, être conservés ou supprimés. Cette suppression réactualisant aussi l'affichage des données.

### 3.3.2 Actualisation des données

La fonction de filtre est liée aux fonctions d'affichage des tableaux. A chaque nouveau filtre, la liste des valeurs est actualisée et utilisée par la fonction d'affichage pour que l'utilisateur visualise, en temps réel, le tableau de données actualisé. Sur l'interface, la longueur du tableau affichée à droite des filtres, nous permet de suivre l'efficacité du filtrage à chaque ajout.

## 3.4 Sélection d'artefact

Lors de l'affichage du tableau, à chaque cellule d'une même ligne est attribué un identifiant unique ainsi que la fonction de sélection grâce à la propriété *onclick*.

Lorsque l'utilisateur clique sur une ligne, la fonction compare l'identifiant avec chaque valeur de la liste contenant le nom des images. Une fois trouvé, il s'assure que cet identifiant n'est pas déjà dans la liste de sélection et l'ajoute ou non à celle-ci.

Est ensuite créé un bloc contenant l'image du fragment et son nom, qui est ajouté au bloc de sélection pour que l'utilisateur puisse le visualiser.

Sur chaque bloc un bouton de suppression est implémenté au cas où l'utilisateur veuille réduire sa sélection. Ce bouton supprime l'identifiant de la liste de sélection et le bloc fragment du bloc de sélection.

## 3.5 Édition des données

Le mode "Edition", activé par un bouton sur l'interface, laisse apparaître les différents boutons des actions possibles.

### 3.5.1 Ajout

Deux possibilités d'ajout ont été programmées. la première par le chargement d'un nouveau fichier de donnée. La conversion du fichier au format XML est effectuée si nécessaire, puis la variable contenant le fichier XML du projet est modifiée pour ajouter les nouvelles lignes. S'en suit la conversion en listes de valeurs et son affichage.

La deuxième possibilité est l'ajout manuel d'une ligne. Pour cela une boucle parcours les différents attributs, et un champs de saisie correspondant à chaque attribut est créé pour entrer la valeur associée. A la validation des ajouts, les valeurs des champs sont successivement ajoutés à une liste. Cette dernière correspond aux données du nouvel artefact. L'artefact est ajouté à la liste originale de valeurs, ce qui permet ainsi de réactualiser l'affichage du tableau. Enfin la variable XML est modifiée pour ajouter cette nouvelle ligne.

### 3.5.2 Modification

Lorsque le bouton est activé, le tableau de données est ré-affiché pour permettre d'appeler la fonction de modification en cliquant sur une des lignes. L'identifiant de la ligne est récupéré et cherché dans la variable XML ainsi que dans la listes de valeurs. De la même façon que pour la fonction d'ajout, des champs de saisie sont créés pour chaque attribut, contenant cette fois les valeurs déjà enregistrées pour qu'elles puissent être modifiées.

Concernant le fichier XML, la ligne spécifique est détectée et modifiée suivant les nouvelles valeurs. L'affichage du tableau est ensuite réactualisé.

### 3.5.3 Suppression

De la même façon que précédemment, l'affichage du tableau est modifié pour relier chaque ligne à la fonction de suppression.

A son activation la ligne correspondante de la variable XML est détectée et supprimée. De même pour la liste des valeurs.

L'affichage du tableau est ensuite réactualisé.

## 3.6 Intégration du code Python

### 3.6.1 Environnement virtuel et modules

La création d'un environnement virtuel dans lequel il sera possible d'installer les différents modules permet d'éviter des problèmes de compatibilité entre les différentes versions de ces modules Python et nos différents projets.

La première étape consiste à créer un répertoire. Dans un terminal depuis ce dossier, il faut ensuite entrer la commande d'activation de l'environnement virtuel :

```
virtualenv -p /usr/bin/python2.7 venv.
```

Pour activer la session de travail sur le projet, il suffira par la suite d'entrer dans un terminal à chaque session :

```
§. venv/bin/activate.
```

Une fois dans l'environnement virtuel, il est nécessaire d'installer les modules Python utiles au bon fonctionnement du programme, soit : Networkx, Tensorflow, Keras, Numpy, OpenCV, SciPy, Matplotlib et enfin Flask.

### 3.6.2 Architecture globale

L'architecture globale des répertoires en utilisant le module Flask est quelque peu modifiée par rapport à l'architecture globale des répertoires sans son utilisation. Le choix d'utiliser un serveur Flask a nécessité de revoir son arborescence. A la racine du projet se trouve à présent l'ensemble des scripts python, le modèle entraîné de l'algorithme d'assemblage (patchAssemble\_ost2.h5), l'ancien répertoire "exemple" et de nouveaux répertoires :

**static et templates** : Dossiers que l'on retrouve dans tous les projets Flask ; ils sont reconnus automatiquement. templates contient l'ensemble des pages html du site web. Dans notre cas il n'y a que la page de l'interface (index.html). Static contient les fichiers intégrés dans les codes html. Nous y avons placé les fichier index.js et index.css, mais également les modules javascript (/lib), et les illustrations de l'interface (/img), placés dans des sous-répertoires distincts pour plus de clarté

**preprocessed\_/test** : Chemin spécifique au script patchAssembly.py. Il contient les résultats de l'algorithme, c'est à dire les graphs correspondants aux propositions d'assemblages, regroupé dans un seul fichier (au format ...).

**venv** : Contient la version Python2.7 et les modules tel que Tensorflow, Flask, Numpy, Keras, Opencv-python, Matplotlib et Networkx qui sont nécessaires au bon fonctionnement de l'algorithme du client et de l'interface. Les modules comme tensorflow et Keras fonctionnent avec des versions antérieures aux versions les plus récentes car certaines méthodes ont été supprimés dans les versions postérieures (Tensorflow 1.13.1 et Keras 2.2.4).

**\_pycache\_** : Répertoire géré automatiquement par le serveur. Il contient des fichiers de caches relatifs aux scripts pythons.

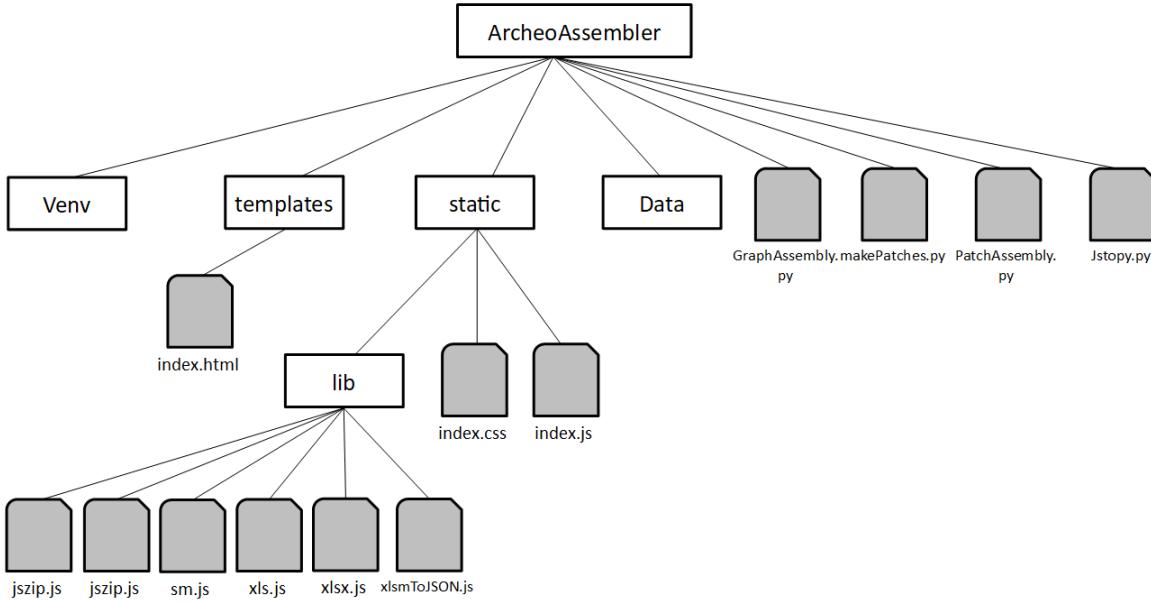


FIGURE 3.1 – Architecture globale du projet avec le module Flask

### 3.6.3 Script jstoppy.py et son fonctionnement

Le serveur en lui-même est un simple fichier python dans lequel le module Flask a été importé ainsi que les scripts de l'algorithme. Toutefois, il possède une structure particulière, avec notamment l'utilisation du décorateur @ qui permet d'enregistrer une fonction et de l'appeler en fonction de certains événements, ici le décorateur `@app.route` permet l'association entre un URL ou/et un argument et une fonction. Ainsi une seule fonction principale agit comme la clef de voûte de l'interface, en contrôlant l'exécution du workflow, en appelant successivement les méthodes et scripts nécessaires.

Pour une application web, il est important que les données puissent être envoyées vers le serveur. On remarque que notre `@app.route` contient un argument appelé `methods=['POST', 'GET']`. L'argument POST permet obtention et l'envoie de données. En implémentant la variable/fonction `panierToFlask()`, nous envoyons, via la méthode POST, les images sélectionnées par l'utilisateur sous le format JSON. Le format JSON est un format ne dépendant d'aucun langage il est donc compris par Python et JavaScript. Grâce à la méthode `request` et l'attribut POST, le serveur est en état de recevoir le panier contenant la sélection d'image sous le format JSON. Par la suite les données sont utilisées et importées dans l'algorithme du client. Cependant quelques modifications ont été apportées au script `makePatches.py`, nous avons englobés une partie du script dans une fonction nommée `ImgToPatch()`, qui prend en argument `name_list` (ici notre panier d'images sélectionnées), nous avons ensuite modifiés le code de manière à ce qu'il compare les images présentes dans le répertoire contenant les images et les images dans `name_list`, ainsi si les images sont présentes dans le dossier et dans la liste alors les images seront convertis au format `tfrecord`. Le reste du script importe les scripts du client et les appelle afin de traiter les patches et d'assembler les graphes.

### 3.6.4 La problématique du binding Python/JavaScript

Un des problèmes majeurs était de trouver un moyen d'utiliser les différents modules nécessaires au bon fonctionnement des scripts de l'algorithme. Beaucoup des autres possibilités évoquées dans la partie conception, ne permettait pas l'importation de scripts avec des modules. Ainsi grâce au serveur Flask, nous pouvons communiquer avec une page HTML/JavaScript avec l'intermédiaire d'un format JSON permettant l'échange de données en local entre les deux langages tout en utilisant les bibliothèques propres à ces langages. En pesant le pour et le contre et bien que l'installation et la manipulation de Flask demande un temps d'adaptation cette solution semble la plus adéquate parmi les différentes possibilités.

## Chapitre 4

# Installation et tutoriel d'utilisation

### 4.1 Pré-requis

L'utilisation de l'interface repose sur les pré-requis suivants :

L'utilisateur répertorie ses données de fouilles dans un fichier de données tabulaires (tableur excel ou équivalent), dans lequel chaque ligne correspond à un artefact, et chaque colonne est associée à un attribut (langue, site de fouille, date, etc...)

Parmi les attributs, un correspond à la clef primaire du tableau et à l'identifiant des images de fragments associés.

L'utilisateur possède un répertoire d'images de fragments pour tout ou partie des artefacts renseignés dans le fichier de données, et les noms attribués sont identiques à l'identifiant.

### 4.2 Créer son environnement virtuel en python2

Cette section ne concerne que les utilisateurs qui possède un système opérateur linux.

Pour créer un environnement virtuel afin accueillir l'interface **ArcheoAssembler**, entrez dans un terminal linux les lignes suivantes :

```
$ mkdir ArcheoAssembler  
$ cd ArcheoAssembler  
$ virtualenv -p /usr/bin/python2.7 venv
```

La commande virtualenv venv crée un dossier dans le répertoire courant qui contient les fichiers exécutables Python, et une copie de la bibliothèque pip. Au lancement du serveur flask, qui sera nécessaire pour faire fonctionner les algorithmes Python, il faudra toujours entrer dans un terminal du répertoire ArcheoAssembler

```
$ source venv/bin/activate
```

afin d'activer l'environnement numérique spécifique au projet ArcheoAssembler. Une fois l'environnement virtuel créé et activé il faut installer les modules nécessaires au bon fonctionnement de l'algorithme en faisant pip install suivi du nom du module. A l'exception de Keras et Tensorflow on peut installer les dernières versions, pour Keras il est préférable d'installer la version 2.2.4 ou inférieur et Tensorflow 1.13.1 ou inférieur.

### 4.3 Installation des fichiers

Une fois les modules installés, les commandes suivantes doivent être entrées dans un terminal :  
*export FLASK\_APP=jstopy.py*

```
export _ENV=development
flask run
```

Si l'installation s'est correctement effectuée le lancement du serveur flask affichera :

```
(venv) owl@owl-Swift-5F314-52G:~/ostracapy2$ export FLASK_APP=jstopy.py
(venv) owl@owl-Swift-5F314-52G:~/ostracapy2$ export FLASK_ENV=development
(venv) owl@owl-Swift-5F314-52G:~/ostracapy2$ flask run
* Serving Flask app "jstopy.py" (lazy loading)
* Environment: development
* Debug mode: on
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 106-338-218
```

FIGURE 4.1 – Lancement du serveur flask

Afin que l'interface fonctionne correctement il est nécessaire déplacer des fichiers dans le répertoire ArcheoAssembler. Au niveau du répertoire courant ArcheoAssembler il est impératif d'avoir les scripts python suivant : graphAssembly.py, jstopy.py, makePatches.py et patchAssembl\_ost2.h5.

Premièrement nous devons créer le répertoire Data avec la commande `mkdir Data` qui accueille le fichier de meta-donnée et les images d'ostraca.

Secondement un répertoire `preprocessed_` qui va recevoir les images d'assemblage

Troisièmement un répertoire `static` qui contient index.css, index.js et sous-répertoire contenant la librairie JavaScript.

Quatrièmement un répertoire `templates` qui contient index.html.

Une fois le serveur lancé, celui-ci tourne en arrière fond, on accède à l'interface en ouvrant un moteur de recherche et en allant à l'adresse indiquée. Ici `http://127.0.0.1:5000`, ce qui correspond à l'adresse locale.

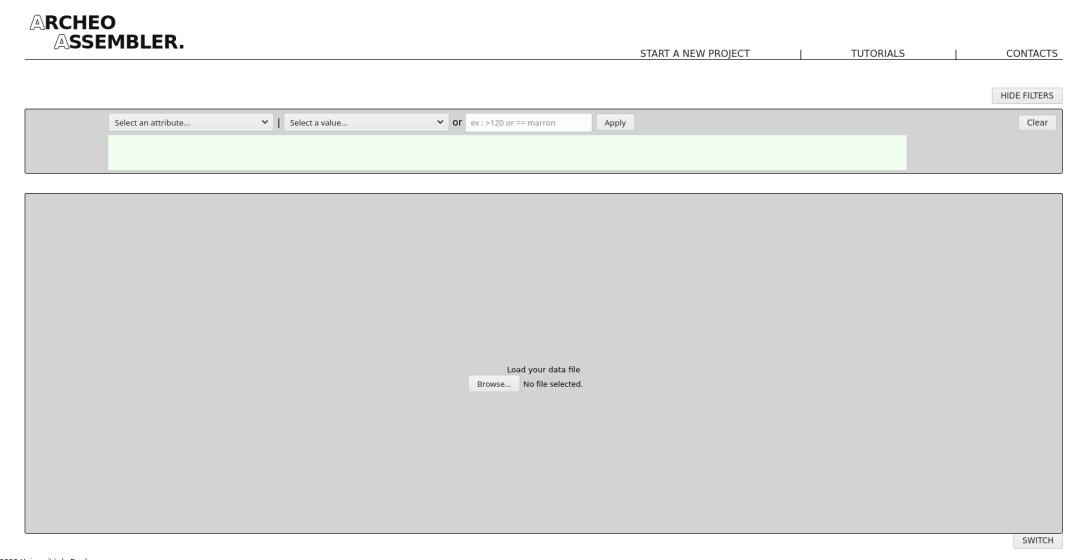


FIGURE 4.2 – Interface client en se rendant sur l'url du serveur local flask

### 4.3.1 Chargement des métadonnées

L'interface web permet de charger des fichiers à partir de dossiers de fichiers locaux. Le format de fichier pris en charge par l'interface web est le .csv qui est ensuite converti en XML permettant la manipulation des données et le filtrage.

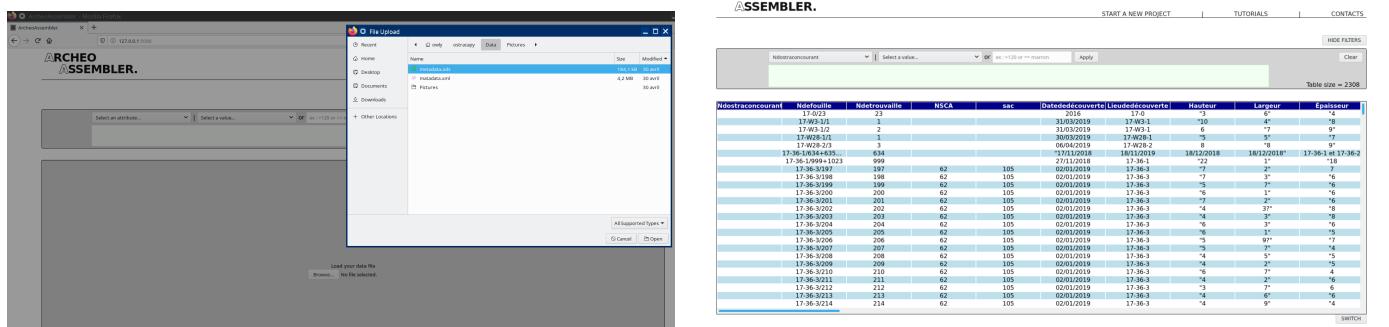


FIGURE 4.3 – Chargement des données avec l’interface web

En appuyant sur le bouton SWITCH on passe de la sélection des meta-données au chargement des images d’ostraca, similairement on va charger des fichiers à partir fichiers contenu dans le dossier contenant les images d’ostraca.

#### 4.3.2 Interactions de l’utilisateur avec l’interface

Si on souhaite sélectionner un ostraca en fonction d’un attribut particulier il suffit de sélectionner l’attribut en question puis de choisir une valeur soit dans le menu déroulant soit en filtrant par la zone de texte.



FIGURE 4.4 – Sélection des ostraca d’intérêts

#### Assemblage d’artefacts

Une fois les fichiers chargés et la sélection d’un ou de plusieurs ostraca, on lance la reconstruction de poteries en appuyant sur le bouton START. Si aucune erreur n’a été faite, le serveur flask va alors lancer les scripts permettant la création de patches et l’assemblage de ceux-ci. Si c’est bien le cas on peut observer dans la console terminale les opérations en cours et l’algorithme qui s’exécute.

#### 4.3.3 Résultats

Lorsque l’algorithme a terminé son assemblage, pour obtenir les résultats il suffit d’aller dans le dossier *ArcheoAssembler/preprocessed\_/test*. Dans ce dossier ce trouvera les hypothèses de reconstructions des ostraca.

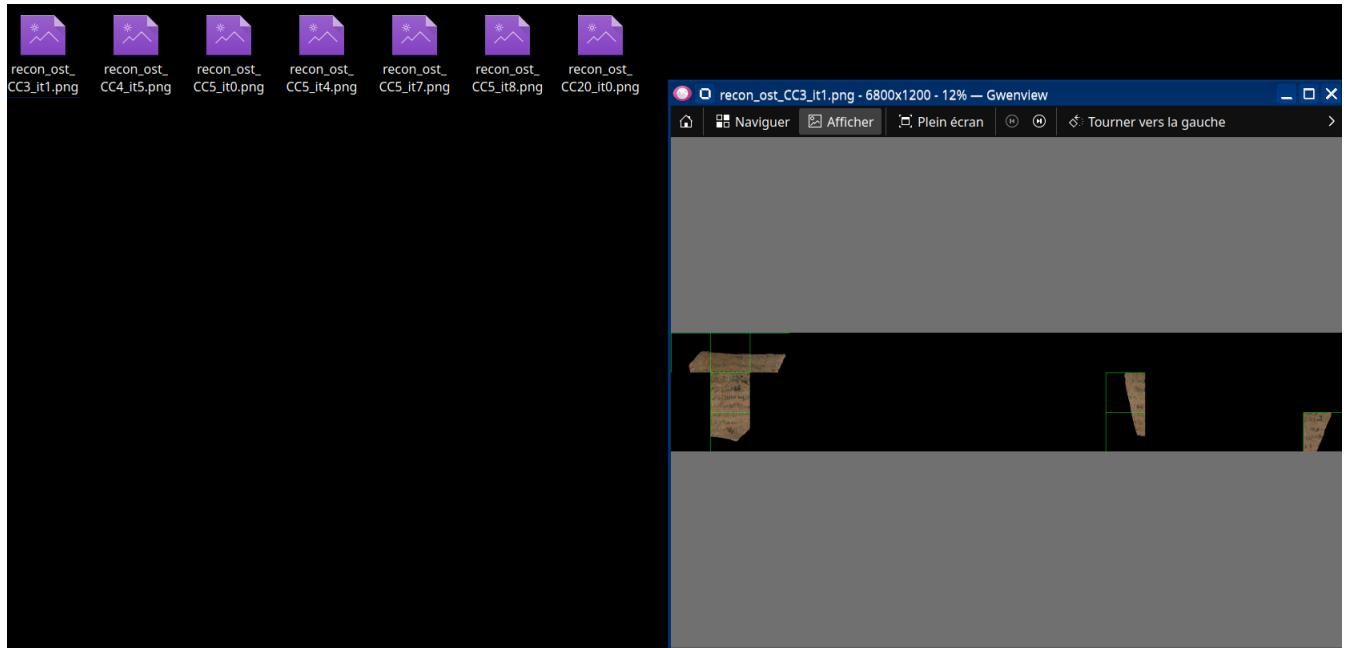


FIGURE 4.5 – Résultats des ostraca d’intérêts

# Conclusion

Le projet ArcheoAssembler a pour but d'intégrer l'algorithme du client dans une interface utilisateur afin de rendre celui-ci accessible pour l'ensemble de la communauté scientifique.

De nombreuses problématiques sont survenues lors de la réalisation de notre interface utilisateur. Notamment la communication entre les deux langages que sont Python et JavaScript. L'intégration des scripts du client a pris plus de temps que prévu. Ils ont été intégrés dans une interface Web qui fonctionne grâce à un serveur local, qui lui-même est exécuté en arrière plan à chaque utilisation. Ceci est relativement peu ergonomique. L'utilisation d'un WSGI permet la communication entre les deux langages mais c'est une implémentation qui exige des connaissances poussées notamment au niveau du module Flask. Une possibilité serait d'utiliser un module Python nommé Tkinter qui permet la création d'un interface visuel pour l'exécution des scripts Python. Ceci réduirait les contraintes de communications entre les différents langages. On peut également noter que l'installation et l'initialisation du serveur local est pour le moment disponible que sous Linux. Il pourrait être judicieux de créer un fichier exécutable avec cx\_Freeze, qui est un module Python, permettant la création de ce fichier à partir des scripts Python.

Pour des raisons de manque de temps et de connaissances pratiques, notre interface Web est fonctionnelle mais imparfaite. Actuellement, lorsque l'utilisateur choisi une ou plusieurs images d'otrac et appuie sur le bouton START, les images sont transformées en patchs et ensuite reconstruites par l'algorithme de ré-assemblage. Une amélioration évidente serait de séparer ces deux scripts (makePatches et graphAssembly) en deux boutons distincts.

Concernant la partie XML de notre code, nous nous sommes rendu compte que nous n'avons pas exploité pleinement les possibilités offertes par le document XML produit.

XSD et XSLT sont des langages couramment utilisés en complément de XML. Le premier langage est utilisé pour établir des règles qui portent sur les valeurs et/ou attributs<sup>1</sup> que peuvent prendre les éléments XML. Le second langage, quant à lui permet d'appliquer des transformations simples (ajout/suppression/édition) ou complexes (définition de fonctions personnalisées), sur tout ou partie du document pour en produire un nouveau. En outre, cet outil offre la possibilité de le convertir un document XML au format XHTML, ce qui aurait permis de l'afficher directement sans avoir à passer par la construction de tableaux.

En effet, notre approche n'est pas la plus optimisée et l'exécution de notre code peut souffrir de ralentissements, notamment lorsque le nombre de filtres appliqués augmente. En l'état, nous avons réussi à limiter ces ralentissements à moins de 10 secondes, mais dans le cas où le fichier utilisateur serait plus volumineux, cela serait susceptible de s'aggraver. D'où l'intérêt de favoriser l'approche qui soit la moins coûteuse en termes d'utilisation des ressources.

A l'issue de ce projet, l'utilisateur a la possibilité de charger ses méta-données, qui sont accessibles à la visualisation par les données brutes (caractéristiques des artefacts, valeurs) mais également par les images représentant les artefacts. Des options de tri, d'édition et de sélection des données sont disponibles. Un regroupement des artefacts sélectionnés par l'utilisateur apparaît à droite de l'écran, donnant lieu à deux actions possibles : la création de patchs dans le cas où une seule image d'artefact aurait été sélectionnée, ou bien une reconstitution 2D à partir de plusieurs images d'artefacts.

Ce projet est une première étape dans la démarche d'accessibilité pour l'ensemble de la communauté scientifique. Il a permis à notre équipe de prendre conscience des différents axes de travail nécessaire à la réalisation d'une interface Web, les contraintes liées à ce projet en respectant un cahier des charges, ainsi que

---

1. A ne pas confondre avec les attributs correspondants aux méta-données.

les différentes possibilités qu'offrent les outils informatiques mis à notre disposition.