

Return 312 : Rapport Technique Projet “OPCI”



Rappel de l'objectif :

Pour répondre à l'appel d'offre, notre groupe Return312 a conçu un logiciel OPCI afin de réduire l'ensemble des vulnérabilités du système interne de l'hôpital en utilisant une combinaison de plusieurs méthodes pour assurer, lors d'un transfert de données, une sécurisation maximale des informations confidentielles des patients. Le logiciel se doit, de plus, d'avoir une utilisation aussi simple qu'agile.

Sommaire

I/ Workflow

II/ Système d'encryption

III/ Base de données

IV/ Serveur

V/ Interface graphique

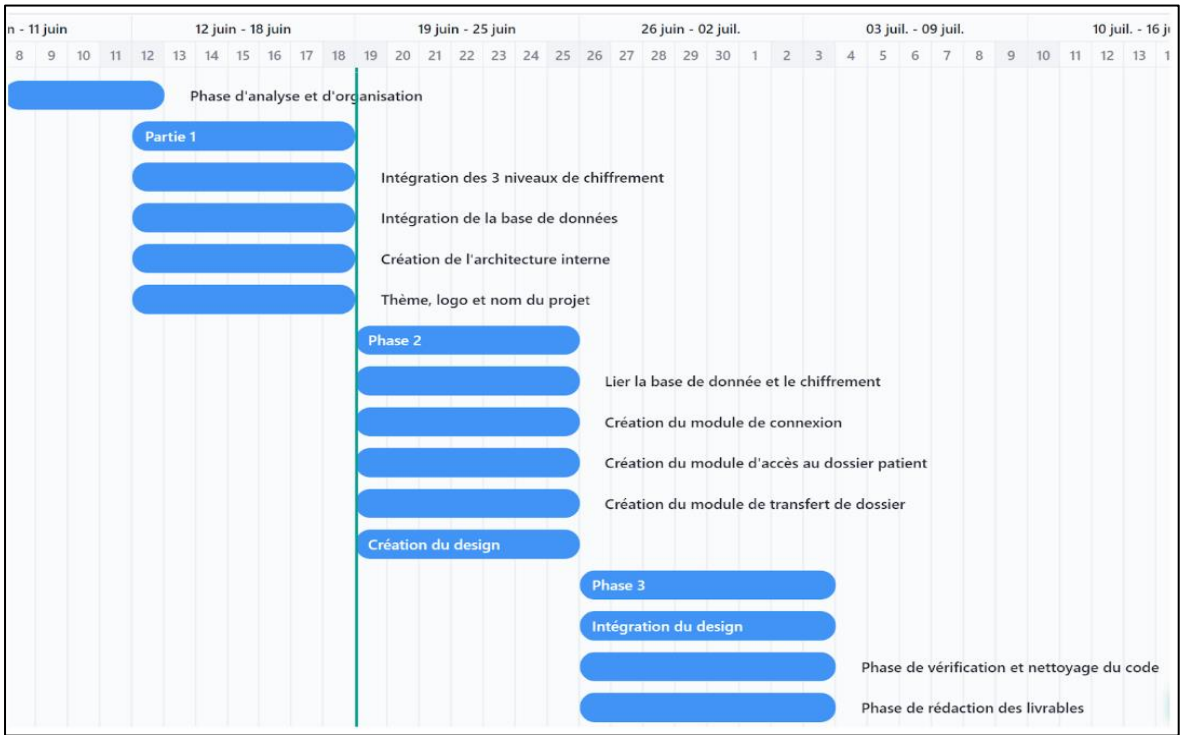
VI/ Fonction de haut niveau et API

VII/ Visualisation de l'architecture du logiciel

I/ Workflow

La réalisation de ce projet a d'abord commencé par la construction des fonctionnalités de bas niveau. Nous avons donc d'abord mis en place le système d'encryption, puis la mise en place de la base de données et de ses fonctions respectives, ensuite nous avons intégré un serveur avec les fonctions nécessaires. Ensuite, nous avons implémenté une API pour lier les parties Front-End et Back-End et à l'intérieur de cette API, nous avons constitué les fonctions de haut niveau qui regroupent l'ensemble des fonctionnalités. Finalement, nous avons intégré le design et les fonctions de notre API pour avoir une interface visuelle.

Le planning de ce projet s'est déroulé de la manière suivante :



II/ Système d'encryption

Le système d'encryption est avant tout la composante majeure du projet car il doit permettre de sécuriser des fichiers à l'aide de différents types de chiffrement.

Ainsi, deux types de chiffrement ont été utilisés pour protéger les fichiers ajoutés à notre solution. Le premier est le chiffrement asymétrique RSA dont la caractéristique est d'identifier les destinataires, qui sont dans notre cas les médecins et les administrateurs, en leur envoyant une clé privée qui leur est spécifique. En outre, ce type d'encryption a nécessité l'utilisation de bibliothèques spécifiques notamment `pyAesCrypt` qui est un module contenant des scripts de chiffrement nécessaire pour pouvoir avoir une compatibilité avec le langage Python.

D'autre part, le chiffrement symétrique AES représente une part non-négligeable du cryptage utilisé dans notre logiciel car il est utilisé pour chiffrer les données de session d'un utilisateur mais aussi pour crypter les données d'un patient. En effet, le chiffrement d'une session nécessite avant tout de crypter les données de fonctionnement qui s'avèrent être sensibles telles que les clés de chiffrement et la liste des patients.

Par ailleurs, le chiffrement des données d'un patient (tel que les fichiers et les informations personnelles qui lui sont associés) sont permis grâce à la clé unique associée au patient et qui est détenue par le médecin. En somme, la sécurité de l'ensemble de notre solution est basée essentiellement sur la sécurité des clés privées.

III/ Base de données

La base de données est un élément essentiel du projet car elle nous permet de stocker les données de chaque médecin : Nom, Prénom, Spécialité, Username, Liste de patients, Clé privée, Clé publique.

```
_id: ObjectId('62c31b6b53169dc33f285c54')
firstname: Binary('QUVTAgAAG0NSRUFURURfQlkAcHlBZXNDcnlv
lastname: Binary('QUVTAgAAG0NSRUFURURfQlkAcHlBZXNDcnlv
speciality: Binary('QUVTAgAAG0NSRUFURURfQlkAcHlBZXNDcnl
privatekey: Binary('QUVTAgAAG0NSRUFURURfQlkAcHlBZXNDcnl
v patients: Object
  Pierre-Dupont: Binary('QUVTAgAAG0NSRUFURURfQlkAcHlBZX
username: "John.Doe"
publickey: "-----BEGIN RSA PUBLIC KEY-----
          MIIBCgKCAQEA15Qr/QCu6iMbKN3t8dkUYGO17+1..."
> messages: Object
```

Représentation d'un utilisateur dans la base de données MongoDB

Nous avons utilisé MongoDB Atlas comme base de données car celle-ci nous permet de stocker des informations sous forme de document (json). Afin de pouvoir effectuer des actions de façon automatisée sur la base de données, nous avons créé plusieurs fonctions en utilisant le package `pymongo` disponible pour le langage Python.

Voici les fonctions principales permettant de gérer la base de données :

get_session(): Récupère les données d'une session utilisateur stockée dans la base de données.

get_all_session(): Récupère la totalité des données stockées sur la base de données

create_session(): Crée une session utilisateur et l'ajoute à la base de données.

delete_session(): Supprime une session utilisateur de la base de données.

update_session(): Met à jour les données d'un utilisateur dans la base de données

add_patient(): Ajoute un patient à la liste de patients d'un utilisateur donné, directement dans la base de données.

delete_patient(): Supprime un patient de la liste de patients d'un utilisateur donné, directement dans la base de données.

IV/ Serveur

Le serveur nous permet de stocker les différents fichiers qui constituent le dossier d'un patient. Nous avons utilisé la technologie Azure Storage pour répondre à ce besoin.

Comme nous utilisons le langage Python, nous avons utilisé le package `azure.storage.blob` lors de la définition de nos fonctions.

```
@classmethod
def upload_file(cls, filepath, upload_filename):
    """
    Function to upload a file to a azure blob storage
    :param filepath: path of the file to upload
    :param upload_filename: name of the uploaded file
    :return: upload_filename, status
    """
    try:
        blob_client = cls.blob_service_client.get_blob_client(
            container=cls.azureContainerName,
            blob=upload_filename
        )
        with open(filepath, "rb") as data:
            blob_client.upload_blob(data, overwrite=True)

        return upload_filename, STATUS.OK
    except Exception as e:
        print("upload_file: " + str(e))
        return None, STATUS.INTERNAL_SERVER_ERROR
```

Aperçu de la fonction `upload_file()`

Voici les principales fonctions permettant de gérer le côté serveur :

`upload_file()`: Télécharge un fichier sur le serveur.

`download_file()`: Télécharge un fichier depuis le serveur.

`delete_file()`: Supprime un fichier sur le serveur.

Pour la réalisation de la maquette de notre application Desktop, nous avons utilisé le serveur Azure pour stocker les données. Toutefois, comme notre application a pour but d'être utilisée sur un serveur local, nous avons implémenté les fonctions du serveur en utilisant un design de pattern stratégique permettant une transition facile vers un serveur local. L'utilisation d'Azure que l'on ne juge pas assez sécurisé est donc purement temporaire.

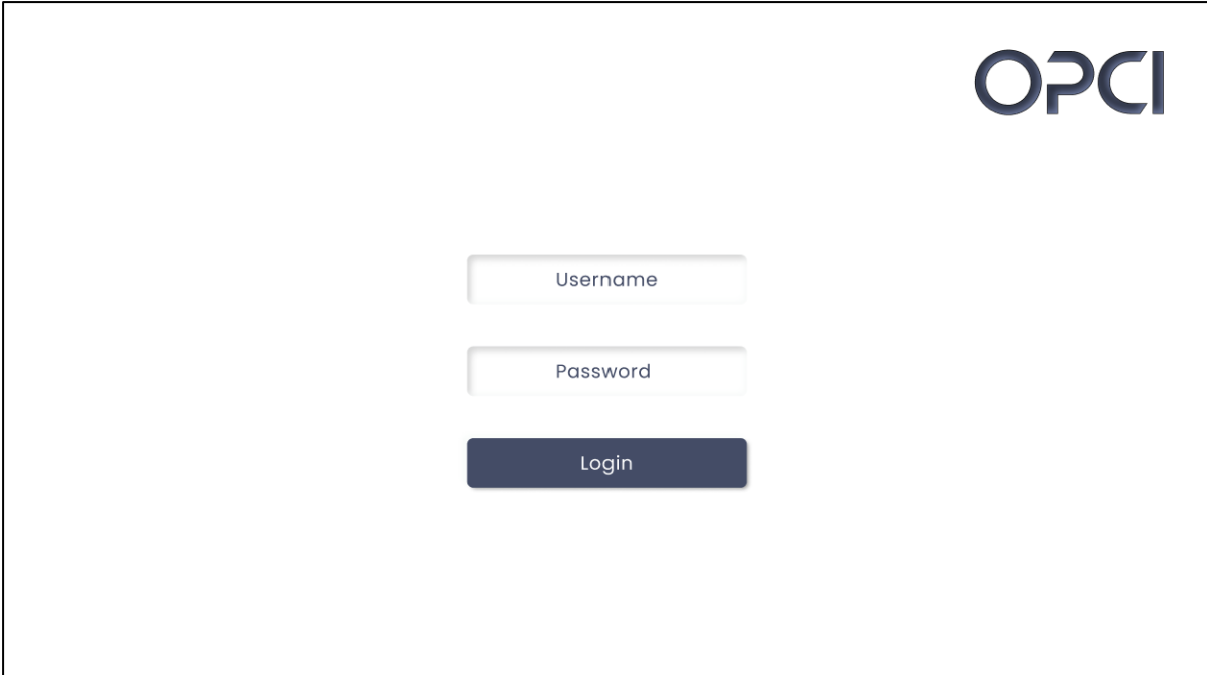
V/ Interface graphique

Après s'être attardé sur le Back-End, il était nécessaire de travailler suffisamment sur le Front-End afin de proposer un logiciel suffisamment intuitif et polyvalent pour la navigation de l'utilisateur, qu'il soit initié ou non. De ce fait, il était donc impératif de travailler en amont sur un modèle à partir de différentes maquettes afin d'avoir un aperçu des différents outils que l'on implémente à partir de notre Back-End.

Ainsi, dans un premier temps, le logiciel Figma s'avérait être l'outil le plus pertinent pour pouvoir réaliser le plus efficacement possible le design de notre logiciel. Une fois accordé sur les différentes maquettes réalisées, notamment sur les couleurs, l'agencement des pages et des images, il était nécessaire d'implémenter efficacement l'ensemble du schéma synoptique conçu en amont :

- Réaliser une animation lors de l'entrée d'un utilisateur sur le logiciel,
- Avoir une interface intuitive pour la page de connexion,
- Mettre en évidence les fonctionnalités d'usage tel que l'ajout d'un dossier sur le drive, la suppression de ce dernier et le partage des fichiers, tout cela de façon sécurisée.

Voici quelques exemples de designs issus de notre logiciel :



The image shows a login page design within a rectangular frame. In the top right corner, the 'OPCI' logo is displayed in a dark blue, sans-serif font. Centered on the page are three input fields: a light gray box with the placeholder text 'Username', a similar light gray box with the placeholder text 'Password', and a solid dark blue box with the text 'Login' in white. The fields are stacked vertically with small gaps between them.

Page de login



Vue d'un dossier patient

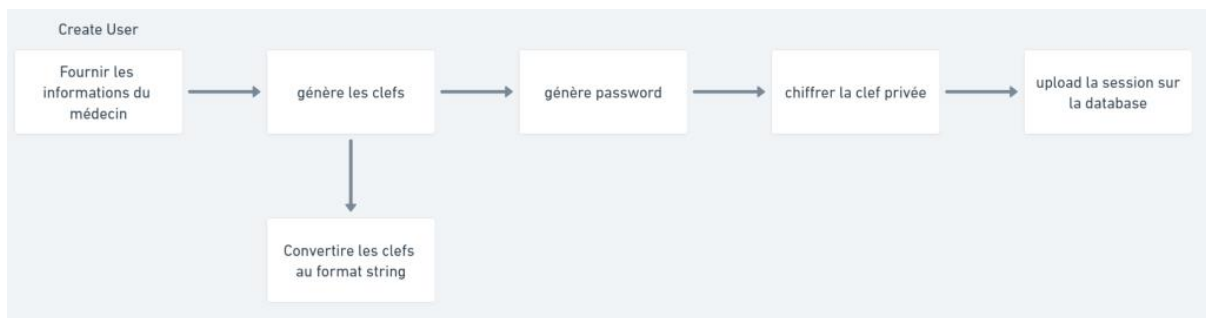
D'autre part, une fois la réalisation des différents scénarios sur Figma, il était impératif de restituer l'ensemble sous un format spécifique. Dans notre cas, nous avons choisis le langage React. Celui-ci nous a permis d'implémenter très facilement l'ensemble des éléments dits « composants » modélisés sur Figma tels que les éléments stylistiques (CSS) ou encore les logos sous format SVG.

VII/ Fonction de haut niveau et API

Comme expliqué précédemment, nous avons créé un logiciel, une application Desktop. Nous avons donc besoin d'intégrer un design pour avoir une interface graphique. Nous avons sélectionné l'API FastAPI comme solution pour cela. Cette API est extrêmement simple à mettre en place et à utiliser, faisant d'elle le choix évident.

Dans notre API nous avons exporté nos différentes fonctions Python pour créer les fonctions majeures de notre application qui regroupent à la fois encryption, base de données et accès serveur. Voici l'organisation des différentes fonctions :

La création d'utilisateur :



Ou sa version en python avec l'appel à fast API `@app.post("/user")`

```

@app.post("/user")
def create_user(newu: User):
    session = newu.dict(exclude={"password"})

    publickey, privatekey = generate_key()
    session['privatekey'] = save_private_key(privatekey)
    session['publickey'] = save_public_key(publickey)
    session['username'] = session['firstname'] + '.' + session['lastname']
    session['patients'] = {}
    session['messages'] = {}
    pwd = newu.password if newu.password else AesEncryption.generate_password()

    print(pwd)

    encrypted_session, error = AesEncryption.encrypt_session(session, pwd)

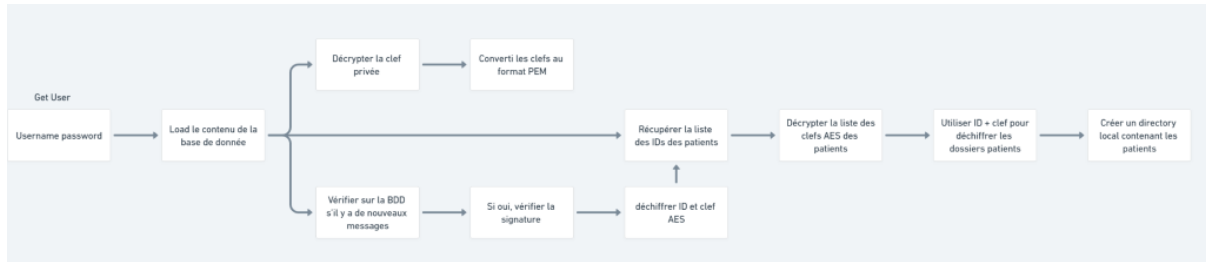
    if error != STATUS.OK:
        return {"message": "Erreur : impossible de crypter la session", "status": STATUS.INTERNAL_SERVER_ERROR.value}

    _, statut = MongoDBDatabase.create_session(encrypted_session)

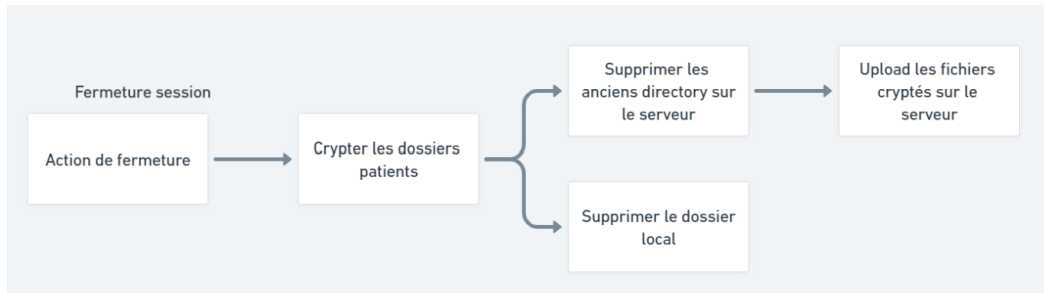
    return {"username": session['username'], "password": pwd, "status": statut.value}
  
```

Return 312

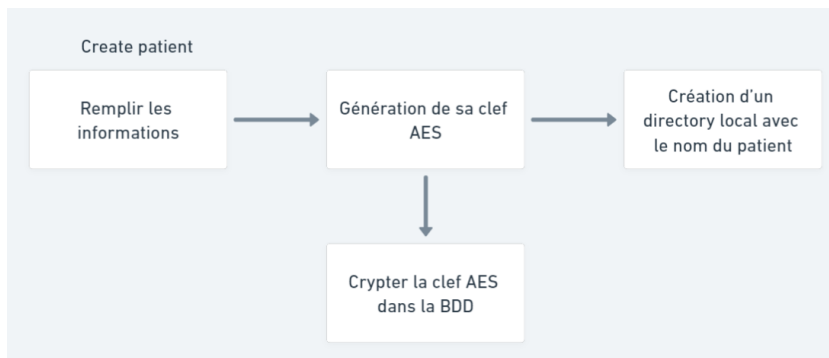
La fonction de connexion :



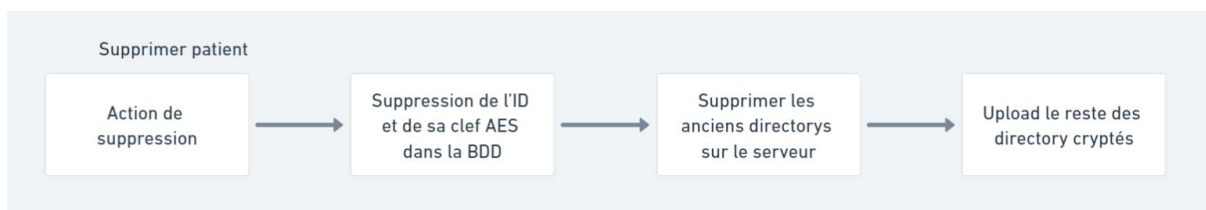
La fermeture de session :



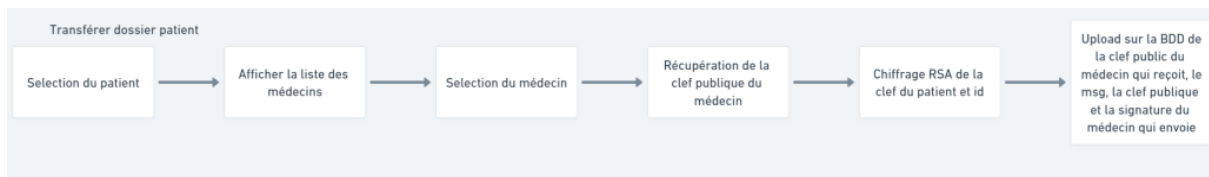
Ajouter un nouveau patient :



Supprimer un patient :



Transférer le dossier d'un patient entre deux médecins :



Une fois toutes ces fonctions implémentées, l'application Desktop est majoritairement prête à l'utilisation.

VII/ Visualisation de l'architecture du logiciel

Voici l'architecture terminée du logiciel sous la forme d'un Diagramme de cas d'utilisation :

