



Département de Mathématiques

Filière Génie Mathématique et Informatique GMI3

Mini Projet Automates – POO

Réalisé par :

Ilhame SADIK
Khadija MOSSAOUI
Nahid ELAIDOUNI

Encadré par :

Pr. AKAMOUISS
Pr. KABIL

Année universitaire
2024/2025

Remerciement

Nous tenons à exprimer notre profonde gratitude à Pr. AKAMOUSS et Pr. KABIL pour leur encadrement bienveillant, leurs conseils précieux et leur disponibilité tout au long de ce projet. Leur expertise et leurs encouragements ont joué un rôle essentiel dans l'aboutissement de ce travail. Nous souhaitons également remercier nos camarades et nos proches pour leur soutien moral et leur compréhension durant cette période de travail intense.

Résumé

Ce mini-projet porte sur la conception, la modélisation et l'implémentation d'un outil interactif pour la gestion des automates. Nous avons développé une application qui permet de créer, manipuler et visualiser des automates finis de manière intuitive. Les fonctionnalités incluent la gestion des états et des transitions, ainsi que des opérations avancées comme la déterminisation, la minimisation et la vérification de mots de passe. Le projet s'appuie sur des concepts de la théorie des automates et est réalisé en utilisant des technologies modernes, avec une interface graphique conviviale facilitant l'interaction avec l'utilisateur.

Abstractf

This mini-project focuses on the design, modeling, and implementation of an interactive tool for finite automata management. We developed an application that enables users to create, manipulate, and visualize finite automata intuitively. Key features include state and transition management, as well as advanced operations such as determinization, minimization, and password verification. The project is based on automata theory concepts and leverages modern technologies to deliver a user-friendly graphical interface that simplifies interaction.

Table des matières

1	Technologie et environnement de développement	8
1.1	Introduction	8
1.2	Langages et outils de programmation	8
1.3	Bibliothèques et Frameworks Utilisés	9
1.4	Conclusion	10
2	Automates et Fonctionnalités Générales	11
2.1	Introduction	11
2.2	Modélisation et Conception	11
2.2.1	Diagramme de classes	11
2.3	Gestion des états	13
2.3.1	Fonction Ajouter un état	13
2.3.2	Fonction Supprimer un état	13
2.3.3	Fonction Modifier un état	13
2.4	Gestion des transitions	13
2.4.1	Fonction Ajouter une transition	13
2.4.2	Fonction Supprimer une transition	14
2.4.3	Fonction Modifier une transition	14
2.5	Opérations sur l'automate	14
2.5.1	Fonction Déterminiser l'automate	14
2.5.2	Fonction Compléter l'automate	14
2.5.3	Fonction Supprimer l'automate	15
2.5.4	Fonction minimiser un automate	15
2.6	Fonction afficher graphe	15
2.7	Interfaces Graphiques pour la Gestion des Automates	15
2.7.1	Gestion des États	15
2.7.2	Gestion des Transition	16
2.7.3	Gestion des Opérations	18
2.7.4	Interface de l'Automate Initiale	18
2.7.5	Automate déterministe	18
2.7.6	Automate complet	20
2.7.7	Automate minimisé	21
2.8	Conclusion	21

3	Automate de Vérification des Mots de Passe	22
3.1	Introduction	22
3.2	Objectifs et Contexte	23
3.2.1	Problématique	23
3.3	Conception	23
3.3.1	Méthodologie et approche adoptée	23
3.3.2	Diagramme de Classe	23
3.4	Implémentation	24
3.4.1	Classe <code>Automate</code>	24
3.4.2	Classe <code>RealTimePasswordApp</code>	26
3.5	Interface de Vérification de Mot de Passe	27
3.6	Cas de Test	28
3.6.1	Cas 1 : Mot de passe faible (lettres minuscules uniquement)	28
3.6.2	Cas 2 :Mot de passe avec majuscules et chiffres, mais sans caractères spéciaux	29
3.6.3	Cas 3 :Mot de passe trop court	30
3.6.4	Cas 4 :Mot de passe trop long	31
3.6.5	Cas 5 :Mot de passe valide avec tous les critères remplis	32
3.6.6	La génération du rapport PDF	33
3.7	Conclusion	35

Table des figures

2.1	diagramme de classe	12
2.2	ajouter etat	16
2.3	supprimer etat	16
2.4	ajouter transition	17
2.5	modifier transition	17
2.6	supprimer transition	17
2.7	Automate Initiale	18
2.8	Automate déterministe	19
2.9	Automate Complet	20
2.10	Automate initiale	21
2.11	Automate minimisé	21
3.1	diagramme de classe	24
3.2	classe automate	25
3.3	classe RealTimePasswordApp	26
3.4	interface utilisateur	28
3.5	Cas 1 : Mot de passe faible	29
3.6	cas 2 :Mot de passe avec majuscules et chiffres, mais sans caractères spéciaux	30
3.7	Cas 3 :Mot de passe trop court	31
3.8	Cas 4 :Mot de passe trop long	32
3.9	Cas 5 :Mot de passe valide	33
3.10	rapport PDF	34

Introduction Générale

Les automates finis sont des outils théoriques et pratiques essentiels dans le domaine de l'informatique. Ils servent à modéliser des systèmes complexes en se basant sur des concepts simples et structurés. Leur utilisation s'étend à divers domaines, tels que la reconnaissance de motifs, la vérification des systèmes, le traitement du langage naturel, ou encore la sécurité des systèmes d'information. Grâce à leur robustesse et leur capacité à formaliser des processus, les automates jouent un rôle clé dans la résolution de problèmes variés, allant de la conception de logiciels aux applications industrielles. Ce rapport explore en profondeur l'univers des automates à travers deux projets principaux :

1. **Le projet général**, axé sur la conception et l'implémentation des différentes fonctionnalités liées aux automates, telles que la gestion des états, des transitions, et les opérations de déterminisation et de complétion.
2. **Le projet spécifique**, l'objectif principal était de développer une solution permettant de valider la sécurité des mots de passe à l'aide d'automates et de techniques de cryptographie. Les automates ont été utilisés pour automatiser et structurer la vérification des critères de sécurité des mots de passe, assurant ainsi une analyse efficace et rapide.

Ces travaux visent à démontrer l'importance des automates comme outils de modélisation et de résolution de problématiques concrètes, tout en mettant en lumière leur potentiel pour répondre aux défis technologiques actuels.

Chapitre 1

Technologie et environnement de développement

1.1 Introduction

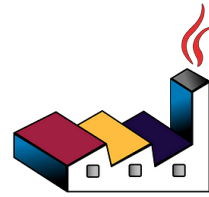
Ce chapitre présente les outils, langages et technologies utilisés pour concevoir et implémenter les projets d'automates. Il décrit également l'environnement de développement choisi, les bibliothèques, et les frameworks employés, ainsi que leur rôle dans la réalisation des objectifs.

1.2 Langages et outils de programmation

- **Python** est un Le langage Python est un langage de programmation open source multi-plateformes et orienté objet. Grâce à des bibliothèques spécialisées, Python s'utilise pour de nombreuses situations comme le développement logiciel, l'analyse de données, ou la gestion d'infrastructures.
- **PyCharm** un éditeur de code complet pour les développeurs Python. PyCharm est un environnement de développement intégré (IDE) utilisé pour la programmation en Python. Il a été développé par JetBrains, une entreprise spécialisée dans les outils de développement logiciel.
- **Git** est un système de contrôle de version distribué, open-source, utilisé pour gérer l'historique et les versions du code source dans des projets de développement. Il permet aux développeurs de collaborer efficacement sur des projets tout en conservant un historique détaillé des modifications.



- **PlantUML** est un outil open-source permettant de créer des diagrammes UML (Unified Modeling Language) à partir de simples descriptions textuelles. Il prend en charge plusieurs types de diagrammes, tels que les diagrammes de classes, de séquence, d'activités, de composants, de cas d'utilisation, et bien d'autres, facilitant ainsi la création rapide et claire de diagrammes dans des projets de développement logiciel.



1.3 Bibliothèques et Frameworks Utilisés

- **Graphviz** est une bibliothèque et un outil logiciel utilisé pour la visualisation et la manipulation de graphes. En Python, elle est accessible via le module graphviz, qui permet de générer des diagrammes de graphes en utilisant des fichiers de description DOT. Elle est particulièrement utile pour représenter des structures comme des automates finis, des arbres de décision, ou des réseaux de dépendances.
- **NetworkX** est une bibliothèque Python puissante et flexible utilisée pour la création, la manipulation, et l'étude de graphes ou de réseaux complexes. Elle est largement utilisée dans les domaines de l'analyse de réseaux sociaux, de la recherche opérationnelle, de la modélisation de systèmes complexes, et bien d'autres.
- **Matplotlib** est une bibliothèque de visualisation de données pour Python qui permet de créer une grande variété de graphiques en 2D (et limitément en 3D).
- **Tkinter** est la bibliothèque standard de Python pour le développement d'interfaces graphiques (GUI - Graphical User Interfaces). Elle permet de créer des applications interactives avec des fenêtres, des boutons, des champs de texte, des menus, et bien d'autres éléments graphiques. Tkinter est basé sur la boîte à outils Tcl/Tk, une bibliothèque multiplateforme pour le développement d'interfaces graphiques.
- **FPDF (Free PDF)** est une bibliothèque Python permettant de générer des fichiers PDF de manière programmatique. Elle offre une solution simple et efficace pour créer des documents PDF directement depuis un script Python, sans dépendance externe lourde.
- **Hashlib** est un module standard de Python permettant de générer des empreintes numériques (hashes) en utilisant différents algorithmes de hachage cryptographique comme MD5, SHA-1, SHA-256, et d'autres. Ces empreintes sont couramment utilisées pour vérifier l'intégrité des données, sécuriser des mots de passe, ou encore générer des signatures numériques.
- **Cryptographie** Bibliothèque pour sécuriser les données via des méthodes de chiffrement, signatures numériques et gestion de clés.

- **Base64** Bibliothèque ou méthode pour encoder/décoder des données binaires en texte lisible (ASCII), utile pour le transfert de données.

1.4 Conclusion

En conclusion, les outils et technologies utilisés, tels que Python, PyCharm, Git, et les bibliothèques comme Graphviz, NetworkX, Matplotlib, et Tkinter, ont permis de créer un environnement de développement efficace et adapté à la conception d'automates. Ces technologies ont facilité la modélisation, la visualisation, la gestion des versions et la création d'interfaces, tout en assurant une mise en œuvre rapide et structurée des fonctionnalités du projet.

Automates et Fonctionnalités Générales

2.1 Introduction

Ce chapitre explore les concepts fondamentaux liés aux automates finis, notamment leur modélisation, conception et manipulation. Un automate est un modèle mathématique utilisé pour décrire des systèmes dynamiques, et il est constitué d'un ensemble d'états, de transitions entre ces états et d'un alphabet de symboles. Ce chapitre présente les principales opérations sur un automate, telles que l'ajout, la suppression et la modification des états et transitions. Il aborde également des techniques avancées comme la complétion et la déterminisation.

2.2 Modélisation et Conception

2.2.1 Diagramme de classes

On considère un automate fini $\text{Aut} = (A, Q, I, T, E)$, où A est l'alphabet, Q l'ensemble des états, $I \subseteq Q$ représente les états initiaux, $T \subseteq Q$ les états finaux, et $E \subseteq Q \times A \times Q$ les transitions. Pour modéliser cet automate, nous adoptons une approche orientée objet basée sur les classes suivantes :

- la classe **État**, qui représente un état de l'automate avec des attributs tels que son identifiant, son type (initial, final ou intermédiaire) et d'autres informations pertinentes ;
- la classe **Alphabet**, qui regroupe les symboles autorisés dans les transitions, utilisés pour définir les conditions de passage entre les états ;
- la classe **Transition**, qui décrit les transitions entre les états à l'aide d'attributs comme l'état d'origine, le symbole déclencheur et l'état de destination ;
- la classe **Automate**, qui représente l'automate dans son ensemble et contient des collections d'états, d'alphabets et de transitions, tout en offrant des méthodes pour manipuler et analyser l'automate.



FIGURE 2.1 – diagramme de classe

2.3 Gestion des états

2.3.1 Fonction Ajouter un état

```
def ajouter_etat(self) -> object
```

La fonction `ajouter_etat` permet à l'utilisateur d'ajouter un nouvel état à l'automate. Elle ouvre une fenêtre modale où l'utilisateur peut entrer l'ID de l'état, ainsi que ses propriétés (état initial et/ou final). Si l'ID de l'état est valide et n'existe pas déjà, l'état est ajouté à l'automate, et la vue graphique de l'automate est mise à jour via la méthode `afficher_graphe`, permettant de refléter ce changement visuellement.

2.3.2 Fonction Supprimer un état

```
def supprimer_etat(self) -> None
```

La fonction `supprimer_etat` demande à l'utilisateur de saisir l'ID de l'état qu'il souhaite supprimer. Si l'état existe dans l'automate, il est supprimé, et la vue graphique est mise à jour avec la méthode `afficher_graphe` pour refléter ce changement visuel après la suppression.

2.3.3 Fonction Modifier un état

```
def modifier_etat(self) -> None
```

La fonction `modifier_etat` permet à l'utilisateur de modifier le nom d'un état existant. L'utilisateur entre l'ID de l'état à modifier, puis un nouveau nom pour cet état. Si l'état existe, son nom est mis à jour, et la vue graphique est rafraîchie via `afficher_graphe` pour afficher l'automate mis à jour. Si l'état n'existe pas, un message d'erreur est affiché.

2.4 Gestion des transitions

2.4.1 Fonction Ajouter une transition

```
def ajouter_transition(self) -> Any
```

La fonction `ajouter_transition` permet à l'utilisateur d'ajouter une nouvelle transition entre deux états. L'utilisateur fournit l'état source, l'état destination, et l'alphabet (symbole ou ensemble de symboles). Si les valeurs sont valides, la transition est ajoutée à l'automate, et la vue graphique est mise à jour via `afficher_graphe`. Cette fonction génère un ID unique pour chaque nouvelle transition.

2.4.2 Fonction Supprimer une transition

```
def supprimer_transition(self) -> None
```

La fonction `supprimer_transition` permet de supprimer une transition existante en fournissant son ID. Après avoir demandé l’ID à l’utilisateur, la transition correspondante est supprimée, et la vue graphique est rafraîchie avec `afficher_graphe` pour refléter la suppression de la transition.

2.4.3 Fonction Modifier une transition

```
def modifier_transition(self) -> None
```

La fonction `modifier_transition` permet de modifier les détails d’une transition existante. L’utilisateur entre l’ID de la transition qu’il souhaite modifier, puis modifie les valeurs de l’état source, de l’état destination, et de l’alphabet. Une fois validée, la transition est mise à jour et la vue graphique est mise à jour via `afficher_graphe`.

2.5 Opérations sur l’automate

2.5.1 Fonction Déterminiser l’automate

```
def transformer_en_deterministe(self) -> object
```

La fonction `transformer_en_deterministe` transforme un automate non déterministe en un automate déterministe. Elle utilise un algorithme de déterminisation pour convertir l’automate, en regroupant les états équivalents et en ajustant les transitions pour en faire un automate déterministe. Après cette opération, la vue graphique de l’automate est mise à jour avec `afficher_graphe`.

2.5.2 Fonction Compléter l’automate

```
def completer_automate(self) -> None
```

La fonction `completer_automate` permet de compléter l’automate en ajoutant un état "puits" et en ajoutant des transitions manquantes entre les états et les symboles de l’alphabet. Si aucune transition n’existe pour un état donné et un symbole donné, une transition vers l’état puits est ajoutée. Cette méthode garantit qu’aucune transition n’est manquante dans l’automate. Après cette opération, la vue graphique est mise à jour via `afficher_graphe`.

2.5.3 Fonction Supprimer l'automate

```
def supprimer_automate(self) -> None
```

La fonction `supprimer_automate` permet de supprimer entièrement l'automate, en réinitialisant les listes des états, des transitions, des états initiaux et des états finaux. Elle rafraîchit ensuite la vue graphique via `afficher_graphe` pour afficher un automate vide, reflétant la suppression de tous les éléments.

2.5.4 Fonction minimiser un automate

```
def minimiser_automate(self): # unknown *
```

La fonction `minimiser_automate` utilise l'algorithme de Moore pour simplifier l'automate. Cet algorithme identifie les états équivalents en regroupant les états ayant un comportement similaire, puis les fusionne. Une fois les états inutiles supprimés et les transitions mises à jour, l'automate est affiché sous sa forme minimale grâce à un rafraîchissement graphique.

2.6 Fonction afficher graphe

```
def afficher_graphe(self, master: {winfo_children}) -> None
```

Cette méthode est utilisée pour afficher la représentation graphique mise à jour de l'automate à chaque fois qu'un changement est effectué, que ce soit lors de l'ajout, de la suppression ou de la modification des états ou des transitions. Elle permet d'assurer que l'affichage de l'automate soit toujours synchronisé avec les modifications effectuées par l'utilisateur.

2.7 Interfaces Graphiques pour la Gestion des Automates

2.7.1 Gestion des États

Dans l'interface de gestion des automates, l'utilisateur dispose de plusieurs fonctionnalités clés pour gérer les états :

- peut ajouter de nouveaux états en entrant l'ID de l'état ainsi que son type (initial ou final). Si aucun type n'est spécifié, l'état est ajouté sans type particulier.

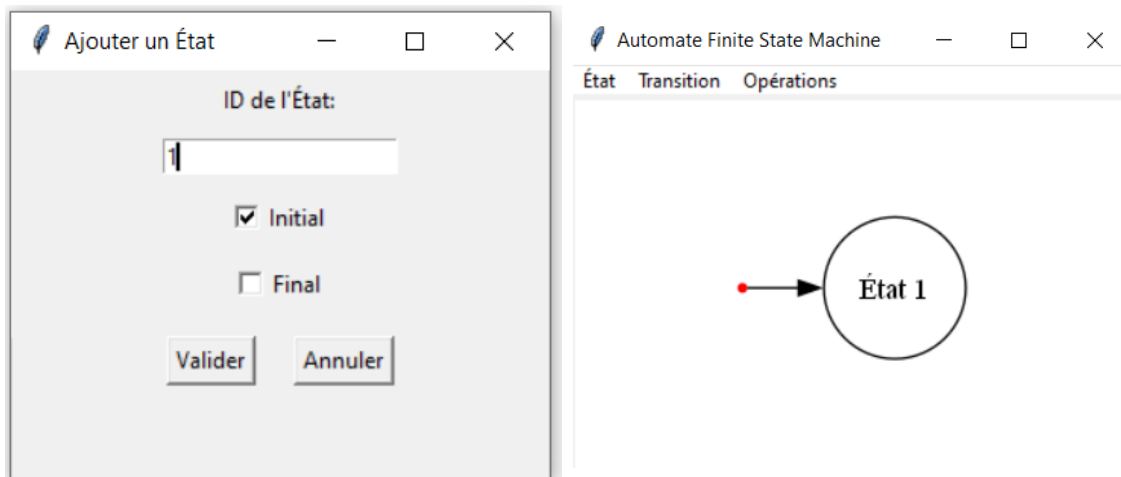


FIGURE 2.2 – ajouter etat

- Il peut également supprimer des états en entrant l'ID de l'état à supprimer dans l'automate..

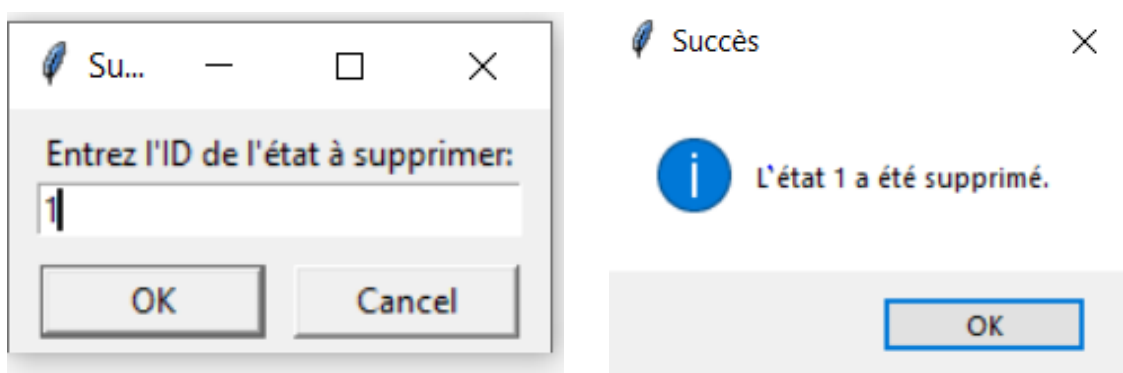


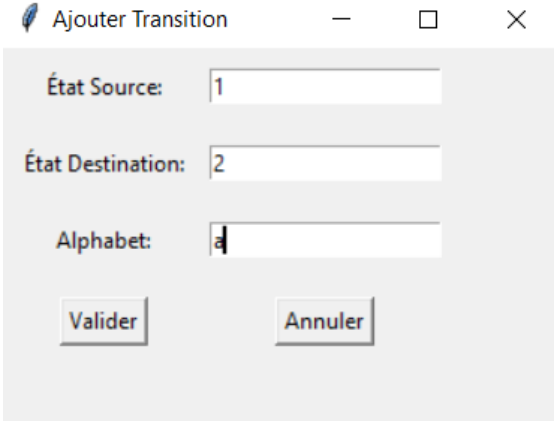
FIGURE 2.3 – supprimer etat

- Il peut également modifier les états existants.

2.7.2 Gestion des Transition

Dans l'interface de gestion des automates, l'utilisateur dispose de plusieurs fonctionnalités clés pour gérer les transitions :

- peut ajouter de nouvelles transitions en spécifiant l'ID de la transition, l'état source, l'état destination, ainsi que l'alphabet associé à la transition. .



Ajouter Transition

État Source: 1

État Destination: 2

Alphabet: a

Valider Annuler

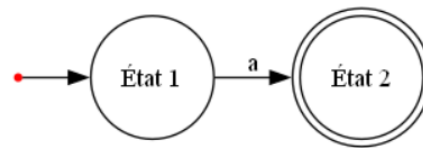
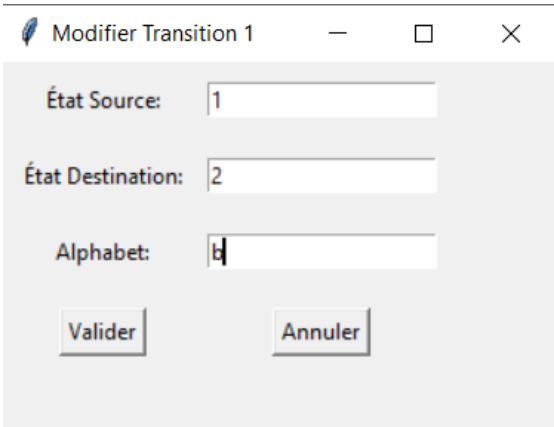


FIGURE 2.4 – ajouter transition

- Il peut également modifier une transition existante en entrant l’ID de la transition à modifier, puis en mettant à jour l’état source, l’état destination et l’alphabet.



Modifier Transition 1

État Source: 1

État Destination: 2

Alphabet: b

Valider Annuler

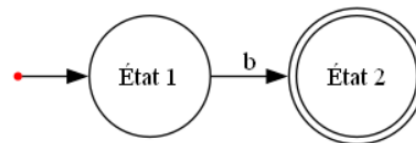
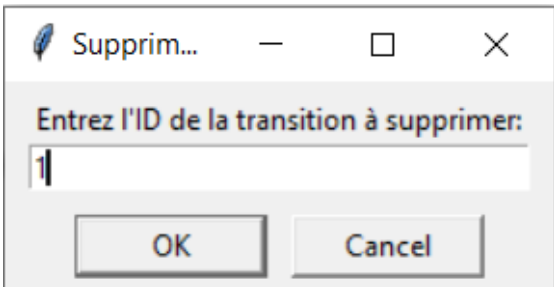


FIGURE 2.5 – modifier transition

- Enfin, l’utilisateur peut supprimer une transition en entrant l’ID de la transition à supprimer dans l’automate.



Supprim...

Entrez l’ID de la transition à supprimer:

1

OK Cancel

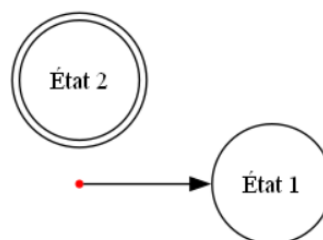


FIGURE 2.6 – supprimer transition

2.7.3 Gestion des Opérations

2.7.4 Interface de l'Automate Initiale

L'automate initial est défini par un alphabet $[a, b, c, d, e]$, un ensemble d'états $[1, 2, 3, 4, 5, 6]$, des états initiaux $[1, 3, 6]$, et un état final $[6]$. Les transitions spécifient les liens entre les états via des symboles, par exemple $(1, a, 2)$ ou $(4, c, 5)$. Cet automate est représenté sous une forme structurée permettant une lecture et une manipulation programmatisées.

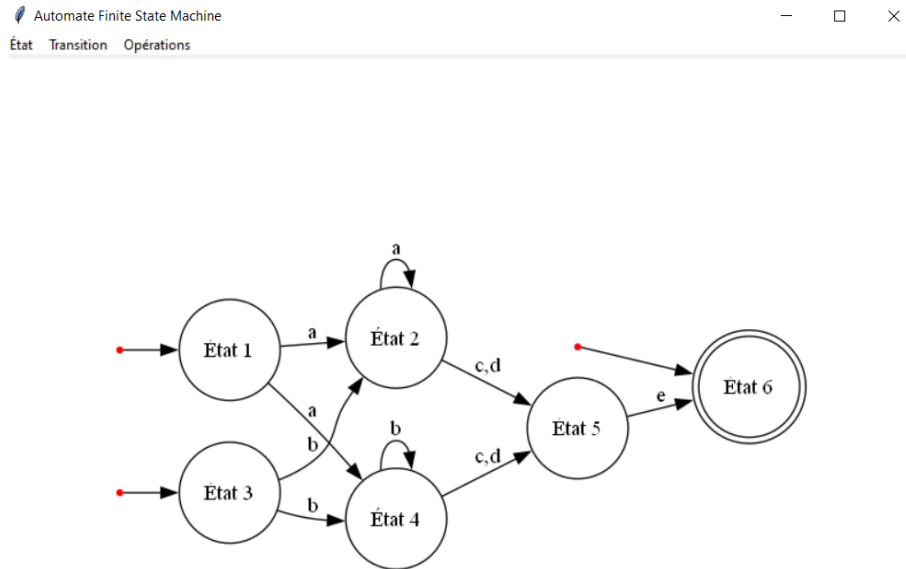


FIGURE 2.7 – Automate Initiale

2.7.5 Automate déterministe

Dans l'interface de gestion des automates, l'utilisateur peut effectuer diverses opérations sur les automates, telles que la déterminisation, la minimisation et la complétion, afin d'optimiser et simplifier les automates.

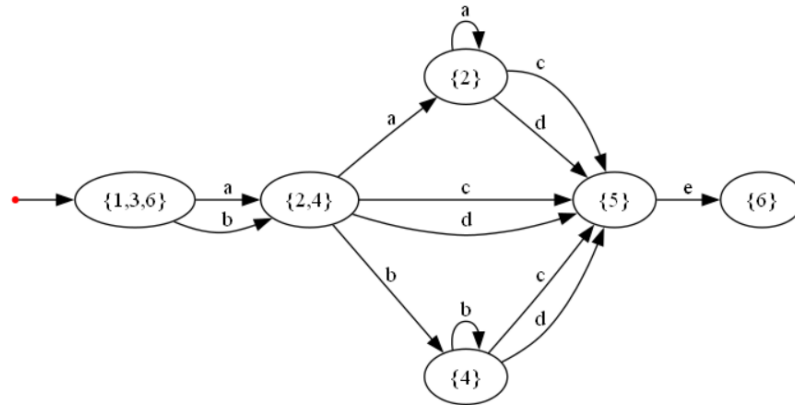


FIGURE 2.8 – Automate déterministe

- **État initial** : Le nouvel automate commence dans l'état déterminisé $\{1, 3, 6\}$, qui regroupe les anciens états 1, 3 et 6 de l'automate non déterminisé.
- **Transitions** :
Chaque transition est associée à un symbole spécifique. Par exemple, la transition $\{1, 3, 6\} \xrightarrow{a} \{2, 4\}$ signifie que, pour le symbole a , l'automate passe de l'état $\{1, 3, 6\}$ à l'état $\{2, 4\}$.
- **État final** : Le nouvel automate déterminisé a deux états finaux : $\{1, 3, 6\}$ (qui contient l'état final original $\{6\}$) et $\{6\}$.

2.7.6 Automate complet

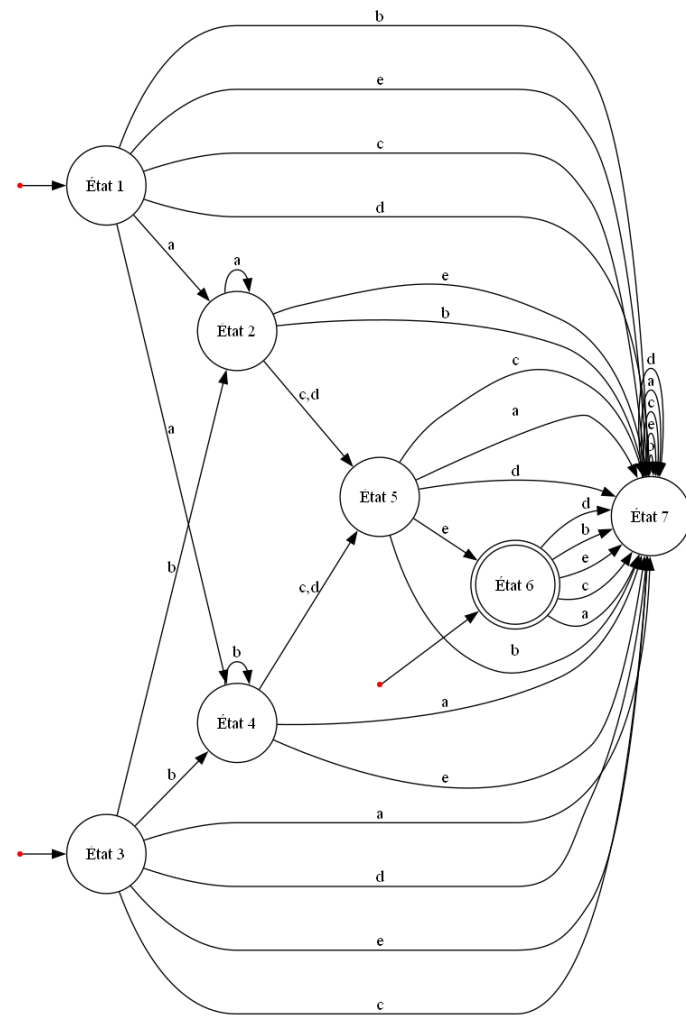


FIGURE 2.9 – Automate Complet

Un nouvel état nommé "poubelle" a été ajouté à l'automate avec l'ID 7. Cet état sert de destination par défaut pour toutes les transitions manquantes. Lorsqu'une transition est attendue mais n'existe pas, l'automate dirige vers cet état "poubelle".

2.7.7 Automate minimisé

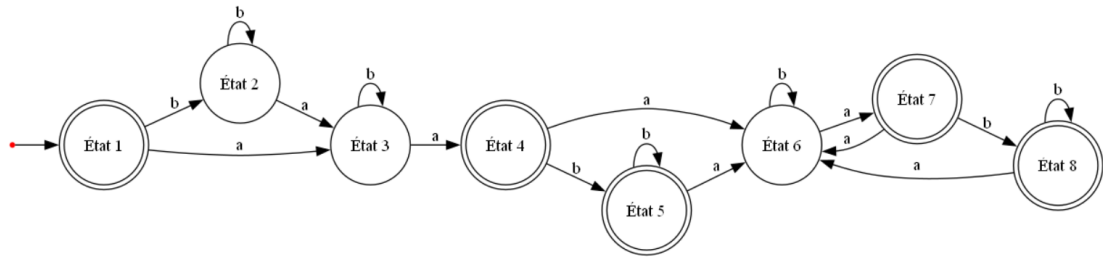


FIGURE 2.10 – Automate initiale

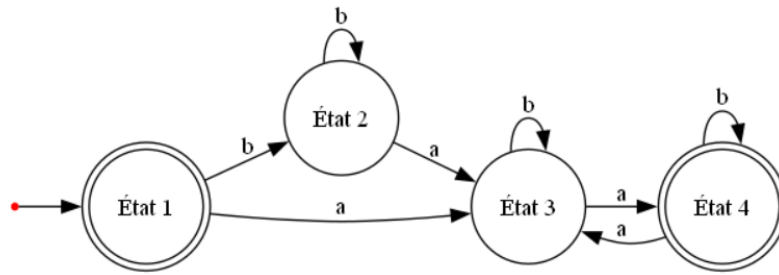


FIGURE 2.11 – Automate minimisé

État 1 = {1}, État 2 = {2}, État 3 = {3, 6}, État 4 = {4, 5, 7, 8}

2.8 Conclusion

En conclusion, ce chapitre présente une approche complète de la gestion des automates, allant de la modélisation UML des classes jusqu'à la conception des fonctions permettant de manipuler les états et transitions. Les différentes opérations sur l'automate, telles que l'ajout, la suppression et la modification des états et transitions, ont été détaillées. De plus, la fonction de complétion de l'automate avec l'ajout d'un état "poubelle" assure que toutes les transitions sont correctement définies. Enfin, les interfaces graphiques facilitent l'interaction avec l'automate, rendant sa gestion plus intuitive et accessible.

Automate de Vérification des Mots de Passe

3.1 Introduction

Dans le cadre de ce projet, l'objectif principal était de développer une solution permettant de valider la sécurité des mots de passe à l'aide d'automates et de techniques de cryptographie. Les automates ont été utilisés pour automatiser et structurer la vérification des critères de sécurité des mots de passe, assurant ainsi une analyse efficace et rapide. Les critères incluent des éléments tels que la longueur du mot de passe, la présence de majuscules, de chiffres et de caractères spéciaux.

Dans ce chapitre, nous explorerons les mécanismes utilisés pour l'analyse des mots de passe, les avantages des automates dans ce processus, ainsi que l'importance de la cryptographie pour la protection des données.

3.2 Objectifs et Contexte

3.2.1 Problématique

Les automates finis sont des outils fondamentaux dans la vérification et la modélisation de systèmes complexes, notamment ceux basés sur des chaînes de caractères. Ce projet vise à exploiter les automates pour vérifier en temps réel la conformité des mots de passe lors de leur saisie dans un formulaire d'inscription.

L'objectif principal est de garantir que les mots de passe saisis respectent des critères de validation spécifiques, notamment :

- La présence d'au moins une majuscule, une minuscule, un chiffre et un caractère spécial.
- Une longueur totale comprise entre 8 et 18 caractères.

Une particularité importante de notre approche est que, durant la saisie du mot de passe, celui-ci est progressivement vérifié par un automate et crypté en temps réel, caractère par caractère, dès que les premières validations sont effectuées. Cela permet de garantir que les mots de passe ne restent jamais en texte clair, même au cours de la saisie.

3.3 Conception

3.3.1 Méthodologie et approche adoptée

Il existe plusieurs langages de modélisation et de conception des modules d'informations. Pour mieux présenter l'architecture de notre module, nous avons choisi la modélisation objet avec le langage UML (Unified Modeling Language).

3.3.2 Diagramme de Classe

Le diagramme de classe est une représentation statique des classes d'un système logiciel et de leurs relations. Il permet de visualiser la structure du système en mettant en évidence les différentes classes, leurs attributs, leurs méthodes et les relations qui les unissent. Ce diagramme est largement utilisé en conception orientée objet pour modéliser la structure des logiciels et faciliter la communication entre les développeurs.

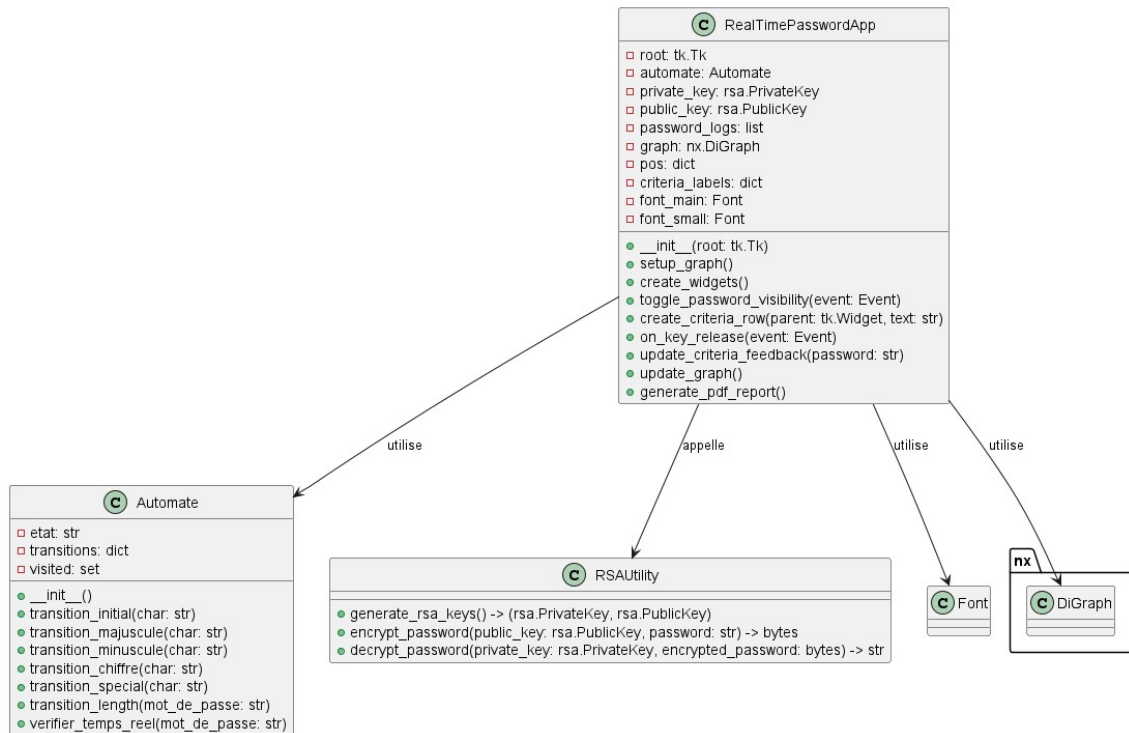


FIGURE 3.1 – diagramme de classe

3.4 Implémentation

3.4.1 Classe Automate

La classe `Automate` est responsable de la validation des mots de passe en temps réel en fonction de plusieurs critères. Cette classe suit un modèle d'automate fini, où chaque caractère du mot de passe fait évoluer l'état de l'automate, en fonction de son type (majuscule, minuscule, chiffre, caractère spécial). L'objectif principal de cette classe est de s'assurer que le mot de passe respecte les critères spécifiés avant d'effectuer des actions supplémentaires.

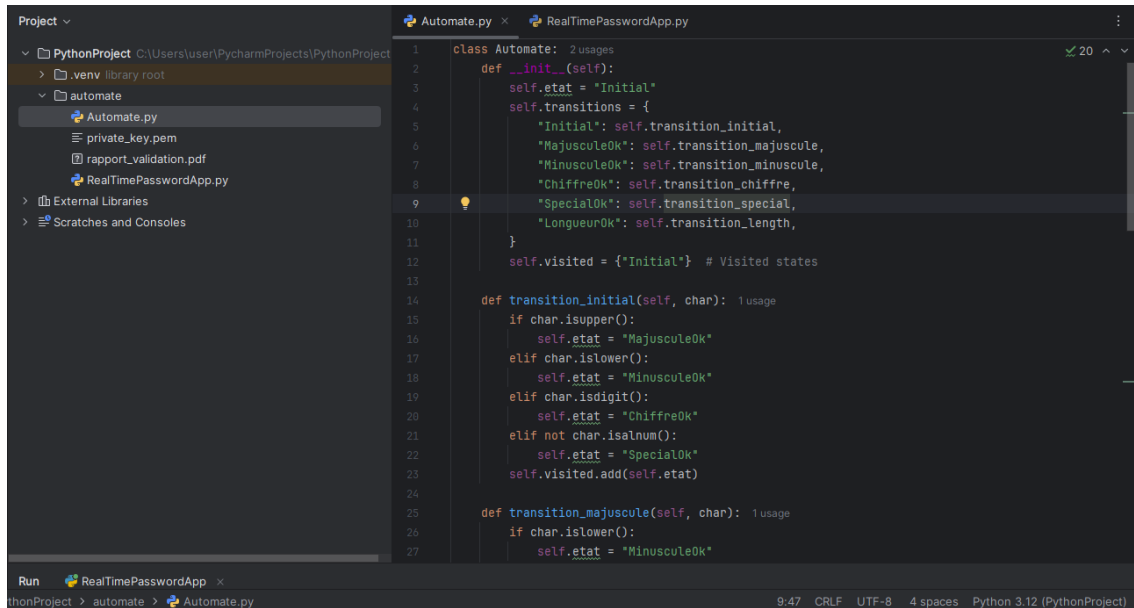


FIGURE 3.2 – classe automate

Attributs

- **etat** : L'état actuel de l'automate (par exemple, "Initial", "MajusculeOk", "MinusculeOk").
- **transitions** : Un dictionnaire qui associe chaque état à une méthode de transition.
- **visited** : Un ensemble qui enregistre les états visités pendant la validation du mot de passe.

Méthodes

Les principales méthodes de l'automate sont :

- **transition_initial(char)** : Gère les transitions à partir de l'état "Initial" selon le type de caractère.
- **transition_majuscule(char)** : Gère les transitions depuis l'état "MajusculeOk".
- **transition_minuscule(char)** : Gère les transitions depuis l'état "MinusculeOk".
- **transition_chiffre(char)** : Gère les transitions depuis l'état "ChiffreOk".
- **transition_special(char)** : Gère les transitions depuis l'état "SpecialOk".
- **transition_length(mot_de_passe)** : Vérifie que la longueur du mot de passe est comprise entre 8 et 18 caractères.
- **verifier_temps_reel(mot_de_passe)** : Méthode principale pour valider le mot de passe en vérifiant chaque caractère et la longueur.

3.4.2 Classe RealTimePasswordApp

La classe `RealTimePasswordApp` est une application interactive permettant la validation en temps réel de mots de passe, la visualisation d'un graphe d'automate représentant les critères de sécurité, et la génération de rapports PDF sécurisés.

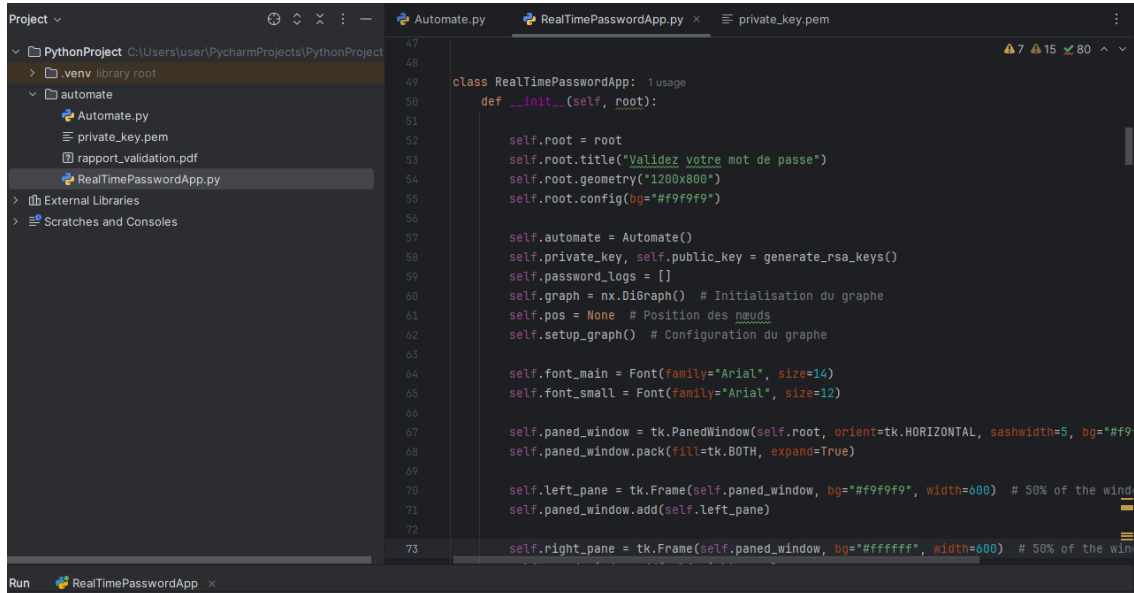


FIGURE 3.3 – classe RealTimePasswordApp

Méthodes

- **`__init__(self, root)`** Initialise l'application, configure l'interface utilisateur, génère les clés RSA et met en place le graphe et les widgets de l'application.
- **`setup_graph(self)`** Configure le graphe d'états pour la validation des critères de mot de passe en utilisant NetworkX.
- **`create_widgets(self)`** Crée les éléments de l'interface utilisateur :
 - Panneau gauche : formulaire utilisateur (nom, prénom, email, mot de passe).
 - Panneau droit : visualisation du graphe des critères.
- **`toggle_password_visibility(self, event=None)`** Permet de basculer entre l'affichage et le masquage du mot de passe saisi.
- **`create_criteria_row(self, parent, text)`** Crée une ligne pour afficher un critère de validation avec un indicateur visuel.
- **`on_key_release(self, event)`** Valide les critères en temps réel à chaque frappe dans le champ de mot de passe :
 - Met à jour les transitions dans le graphe.
 - Ajoute les logs des validations.

- **update_criteria_feedback(self, password)** Met à jour les indicateurs visuels pour chaque critère et affiche le niveau de sécurité du mot de passe (faible, moyen, fort).
- **update_graph(self)** Actualise la visualisation du graphe pour refléter les états atteints selon les critères validés.
- **generate_pdf_report(self)** Génère un rapport PDF contenant :
 - Les informations utilisateur (nom, prénom, email).
 - Les mots de passe testés (chiffrés avec RSA).
 - Les transitions associées à chaque mot de passe.

Méthodes auxiliaires

1. **generate_rsa_keys()** Génère une paire de clés RSA pour chiffrer les mots de passe.
2. **encrypt_password(public_key, password)** Chiffre un mot de passe avec une clé publique RSA.
3. **decrypt_password(private_key, encrypted_password)** Déchiffre un mot de passe avec une clé privée RSA.

3.5 Interface de Vérification de Mot de Passe

Voici notre interface de vérification de mot de passe :

- L'interface permet de tester les mots de passe et de valider leur sécurité en fonction des critères définis.
- Elle est divisée en deux parties :
 1. **Formulaire d'inscription** : Permet à l'utilisateur d'entrer ses informations personnelles, à savoir :
 - Prénom,
 - Nom,
 - Email,
 - Mot de passe.
 2. **Évaluation en temps réel** : Affiche les critères remplis et non remplis. Cette partie comprend :
 - Une liste textuelle des critères de validation (présence de majuscules, chiffres, caractères spéciaux, etc.),
 - Un graphe interactif qui visualise les nœuds et transitions valides selon le mot de passe saisi.

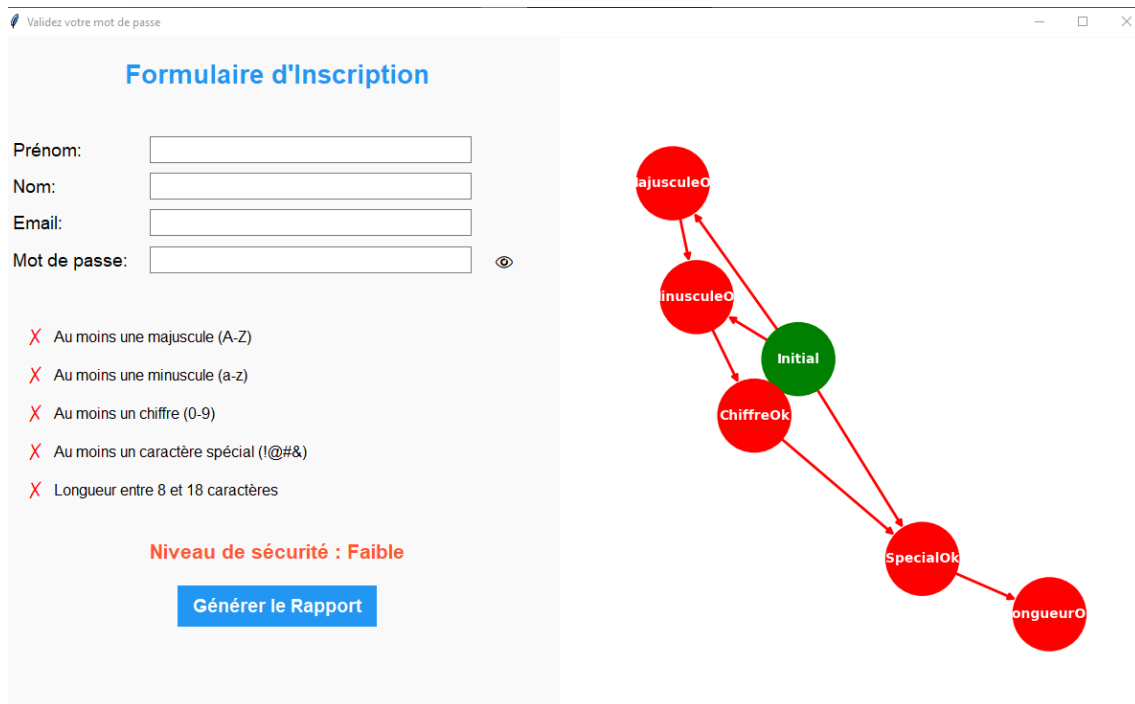


FIGURE 3.4 – interface utilisateur

3.6 Cas de Test

Les tests ont pour objectif de vérifier :

- La validation correcte des mots de passe en temps réel.
- Le fonctionnement du graphe représentant les transitions.
- La génération correcte d'un rapport PDF contenant les informations utilisateur et les mots de passe chiffrés.

3.6.1 Cas 1 : Mot de passe faible (lettres minuscules uniquement)

Vérifier que le système détecte correctement un mot de passe faible ne contenant que des lettres minuscules, sans majuscules, chiffres, caractères spéciaux, ni longueur adéquate.

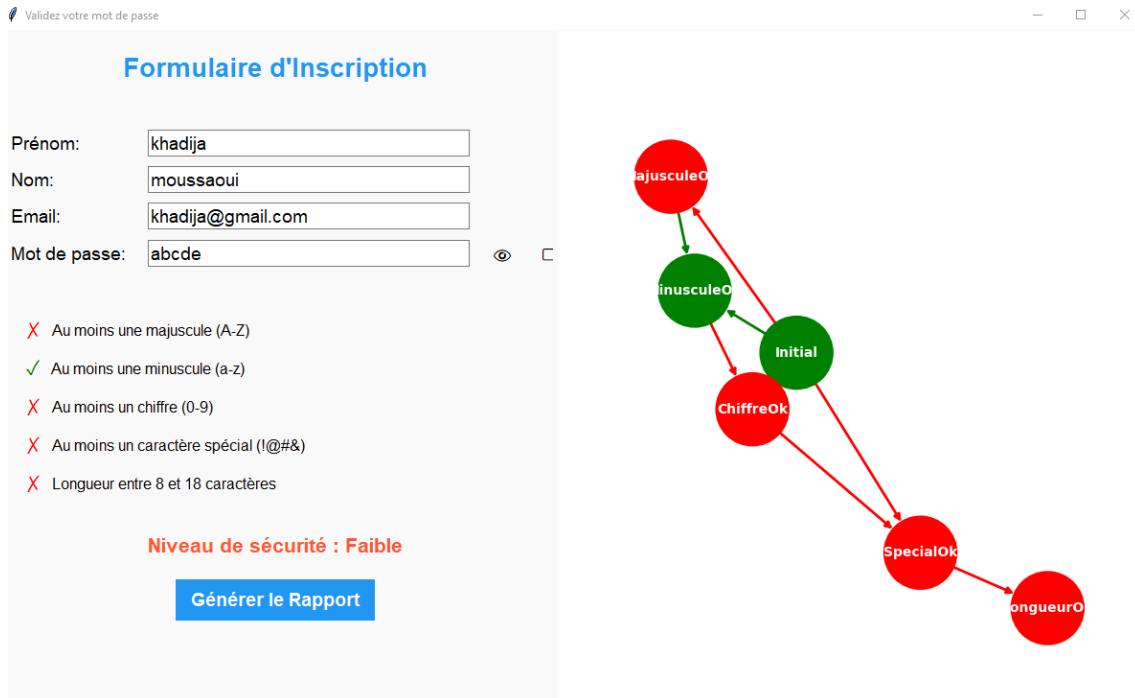


FIGURE 3.5 – Cas 1 : Mot de passe faible

- Cas 1 : Mot de passe testé “abcde”.
- Critères remplis : lettres minuscules.
- Critères manquants : majuscules, chiffres, caractères spéciaux, longueur suffisante.
- États activés : Initial, minusculeOk.
- États inactifs : majusculeOk, chiffreOk, specialOk, longueurOk.
- Le mot de passe est faible car il ne satisfait qu’un seul critère.

3.6.2 Cas 2 : Mot de passe avec majuscules et chiffres, mais sans caractères spéciaux

Vérifie qu’un mot de passe avec majuscules et chiffres mais sans caractères spéciaux est détecté comme insuffisant.

Validez votre mot de passe

Formulaire d'Inscription

Prénom:

Nom:

Email:

Mot de passe: ☐

- ✓ Au moins une majuscule (A-Z)
- ✓ Au moins une minuscule (a-z)
- ✓ Au moins un chiffre (0-9)
- ✗ Au moins un caractère spécial (!@#&)
- ✓ Longueur entre 8 et 18 caractères

Niveau de sécurité : Moyen

[Générer le Rapport](#)

```

graph TD
    majusculeOk((majusculeOk)) -- vert --> Initial((Initial))
    minusculeOk((minusculeOk)) -- vert --> Initial
    Initial -- vert --> ChiffreOk((ChiffreOk))
    ChiffreOk -- rouge --> SpecialOk((SpecialOk))
    SpecialOk -- rouge --> longueurOk((longueurOk))
  
```

FIGURE 3.6 – cas 2 :Mot de passe avec majuscules et chiffres, mais sans caractères spéciaux

- Cas 2 : Mot de passe testé “Abc12345”.
- Critères remplis : majuscules, chiffres.
- Critères manquants : caractères spéciaux, longueur suffisante.
- États activés : Initial, majusculeOk, chiffreOk.
- États inactifs : specialOk, longueurOk.
- Le mot de passe est un niveau de sécurité moyen car il manque des caractères spéciaux et la longueur suffisante.

3.6.3 Cas 3 :Mot de passe trop court

Vérifie qu’un mot de passe ne respectant pas la longueur minimale est rejeté.

Validez votre mot de passe

Formulaire d'Inscription

Prénom:

Nom:

Email:

Mot de passe: ☐ ☐

- ✓ Au moins une majuscule (A-Z)
- ✓ Au moins une minuscule (a-z)
- ✓ Au moins un chiffre (0-9)
- ✓ Au moins un caractère spécial (!@#&)
- ✗ Longueur entre 8 et 18 caractères

Niveau de sécurité : Moyen

[Générer le Rapport](#)

```

graph TD
    Initial((Initial)) --> majusculeOk((majusculeOk))
    Initial --> minusculeOk((minusculeOk))
    Initial --> ChiffreOk((ChiffreOk))
    Initial --> SpecialOk((SpecialOk))
    majusculeOk --> minusculeOk
    minusculeOk --> ChiffreOk
    ChiffreOk --> SpecialOk
    SpecialOk --> longueurOk((longueurOk))
  
```

FIGURE 3.7 – Cas 3 :Mot de passe trop court

- Cas 3 : Mot de passe testé “A1!a”.
- Critères remplis : majuscules, chiffres, caractères spéciaux.
- Critères manquants : longueur suffisante.
- États activés : Initial, majusculeOk, chiffreOk, specialOk.
- États inactifs : longueurOk.
- Le mot de passe est rejeté car il ne respecte pas la longueur minimale requise.

3.6.4 Cas 4 :Mot de passe trop long

Vérifie qu’un mot de passe dépassant la longueur maximale est rejeté.

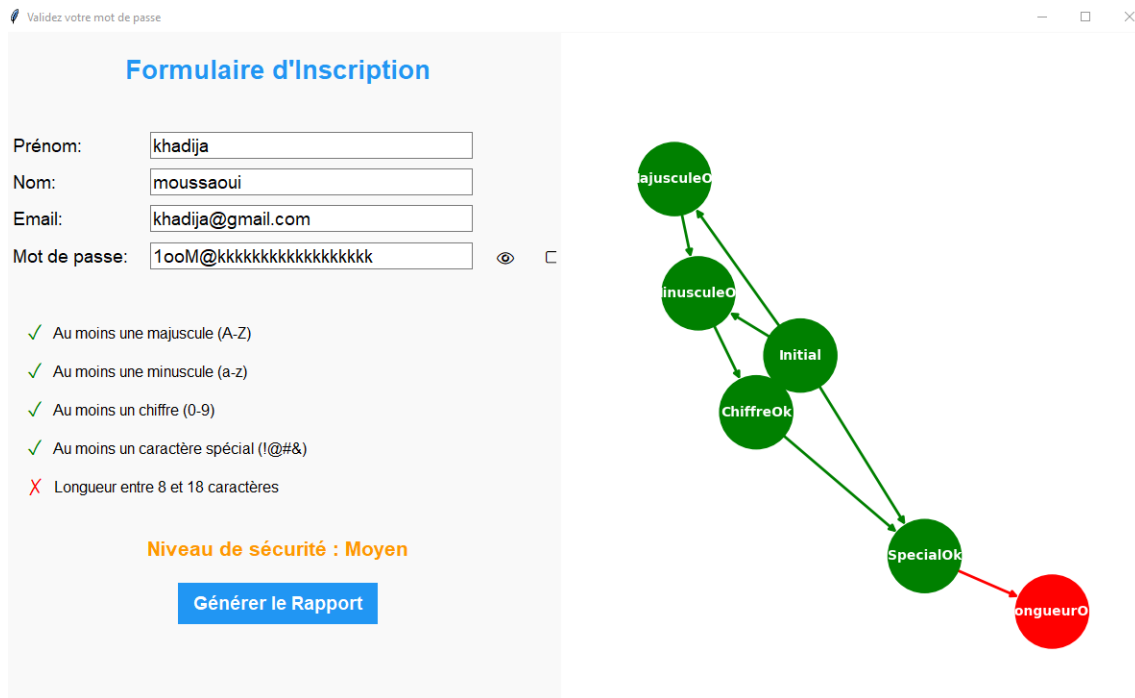


FIGURE 3.8 – Cas 4 :Mot de passe trop long

- Cas 4 : Mot de passe testé : 1ooM@kkkkkkkkkkkkkkkkkkkk.
- Critères remplis : majuscules, chiffres, caractères spéciaux, longueur minimale.
- Critères manquants : aucun (le mot de passe satisfait tous les critères de sécurité).
- États activés : Initial, majusculeOk, chiffreOk, specialOk, longueurOk.
- États inactifs : Aucun.
- Le mot de passe est rejeté car il dépasse la longueur maximale autorisée.

3.6.5 Cas 5 :Mot de passe valide avec tous les critères remplis

Vérifie que tous les critères sont correctement validés pour un mot de passe fort.

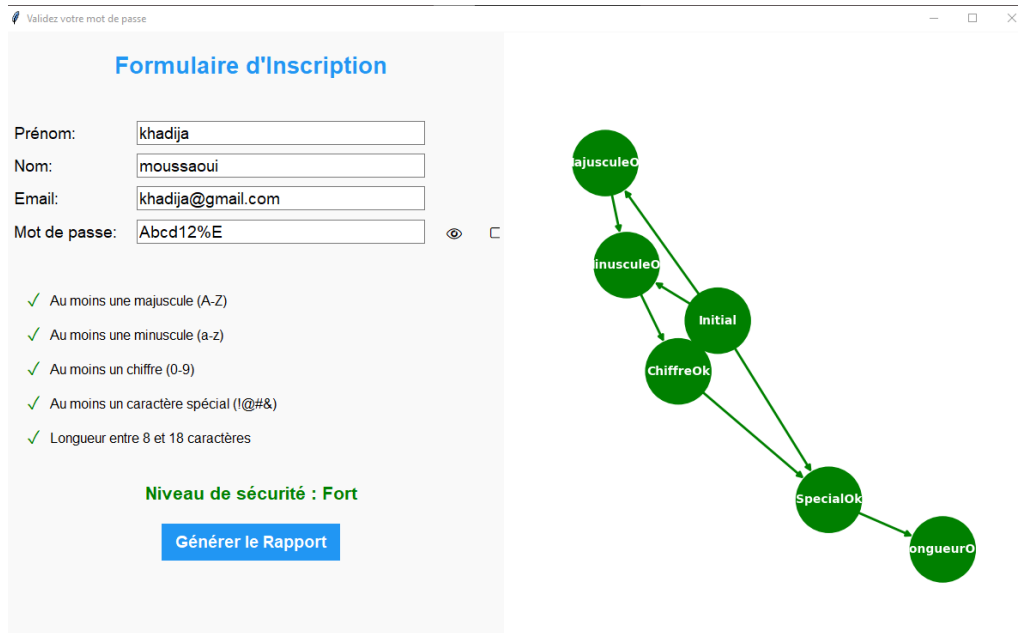


FIGURE 3.9 – Cas 5 :Mot de passe valide

- Cas 5 : Mot de passe testé Abcd12
- Critères remplis : majuscules, chiffres, caractères spéciaux, longueur suffisante.
- Critères manquants : Aucun.
- États activés : Initial, majusculeOk, chiffreOk, specialOk, longueurOk.
- États inactifs : Aucun.
- Le mot de passe est valide car tous les critères de sécurité sont remplis.

3.6.6 La génération du rapport PDF

L'objectif de ce rapport est de fournir un document PDF avec les informations suivantes :

- **Informations utilisateur** : Le rapport inclut des données essentielles sur l'utilisateur, telles que l'identifiant ou le nom d'utilisateur, pour lier chaque mot de passe testé à l'utilisateur correspondant.
- **Mots de passe chiffrés** : Les mots de passe sont hachés ou cryptés afin de garantir leur confidentialité et leur sécurité. Ils ne sont jamais affichés en clair dans le rapport.
- **Critères de validation** : Le rapport présente les critères de validation des mots de passe (longueur, majuscules, chiffres, caractères spéciaux) et indique si chaque mot de passe est validé ou rejeté.

Voici le rapport généré : il contient les informations utilisateurs, les mots de passe analysés et l'évaluation de leur conformité aux critères de sécurité.

Rapport de Validation des Mots de Passe

Nom : moussaoui

Prénom : khadija

Email : khadija@gmail.com

Mot de passe (crypté): ShMBkPalygrlbfKOU07li6P3kj1jV7NxE/i+fAaMLOQieKLqweq41JdgiVK13BAvy13H

Transitions :

MajusculeOk: Valide

Initial: Non Valide

Mot de passe (crypté): DDoXXeMgz9fwW6o5E0lBYWO2OWqWV2dwgd5Dd7KhPvECpsgYcTsEluomgs4l

Transitions :

MajusculeOk: Valide

Initial: Non Valide

Mot de passe (crypté): ek+7JRtljmiG0AQh/V3Pv++0qB2JTEq/2KJMwVU32Ux/BZ+udOVqx58DdNXawAz

Transitions :

Initial: Non Valide

Mot de passe (crypté): eekpJSHEq3Ccu9c+Ci716EjQT0aFwtFePWmGgY8WwTTMK3R3bo/F9VKgKGZQ

Transitions :

MinusculeOk: Valide

Initial: Non Valide

Mot de passe (crypté): cW3WGzo+a4QPYLNNVOUObjKC27v1hViDdLTr8H+oLEV8+CpvD7rLXFSnGKGI

Transitions :

MinusculeOk: Valide

FIGURE 3.10 – rapport PDF

3.7 Conclusion

Ce projet a montré l'efficacité des **automates** pour valider les critères de sécurité des mots de passe, en automatisant leur vérification selon des règles définies (longueur, majuscules, chiffres, caractères spéciaux). L'automate permet une validation structurée et rapide des mots de passe, garantissant la conformité avec les exigences de sécurité.

Parallèlement, des techniques de **cryptographie** ont été utilisées pour protéger les mots de passe, les rendant illisibles même en cas de violation de données. Cette combinaison d'automates pour la validation et de cryptographie pour la sécurisation des données assure une gestion efficace et sécurisée des mots de passe.

Conclusion Générale

en conclusion, ce travail a permis de développer deux projets distincts mais complémentaires. Le premier projet a présenté une approche complète de la gestion des automates, depuis la modélisation UML des classes jusqu'à la conception des fonctions permettant de manipuler les états et transitions. Les différentes opérations, comme l'ajout, la suppression et la modification des états et transitions, ont été détaillées. De plus, la fonction de complétion de l'automate, intégrant un état "poubelle", garantit une définition exhaustive des transitions. Les interfaces graphiques associées rendent cette gestion plus intuitive et accessible.

Le second projet s'est concentré sur l'utilisation des automates pour valider les mots de passe selon des critères de sécurité bien définis (longueur, majuscules, chiffres, caractères spéciaux). En automatisant la vérification, il assure une validation structurée et rapide. En complément, des techniques de cryptographie ont été mises en œuvre pour sécuriser les mots de passe, les rendant illisibles même en cas de violation des données. Cette combinaison d'automates pour la validation et de cryptographie pour la sécurisation garantit une gestion robuste et fiable des mots de passe. Ensemble, ces projets démontrent la pertinence des automates dans des applications variées, qu'il s'agisse de modélisation ou de sécurité informatique.