Universidad de Valladolid

Escuela de Ingeniería Informática de Valladolid

Aplicación web para creación, compartición y simulación de autómatas

Trabajo de Fin de Grado
Grado en Ingeniería Informática
Mención: Computación

Autor: Adrián Sebastián Cantarino

Tutor: Valentín Cardeñoso Payo

Junio de 2022

Agradecimientos

Me gustaría agradecer a mi familia por todo el apoyo desde que empecé la carrera y por ayudarme en los momentos difíciles y complicados.

También a mis amigos por estar siempre ahí cuando hacía falta.

Y por último, pero no menos importante, a los docentes del Grado de Informática de la UVa, en especial a Valentín Cardeñoso Payo, tutor de este trabajo, sin el cuál no se hubiera llevado a cabo.

Resumen

El trabajo describe el desarrollo de una aplicación web AutoShare, que permite crear y simular autómatas, compartirlos con otros usuarios que usan la aplicación y que ofrece persistencia de dichos autómatas en una base de datos.

Se ha utilizado Javascript para desarrollar la aplicación, utilizando React para la parte del Front-End y Node para la parte del BackEnd y PostgreSQL para la base de datos.

Índice general

Índice de tablas

Índice de figuras

Parte I

Objeto, Concepto y Método

Capítulo 1

Introducción y Objetivos

1.1. Introducción

En este Trabajo de Fin de Grado (TFG) se realiza una aplicación web que permita la simulación de autómatas finitos y autómatas de pila, y pueda mostrar paso a paso cómo dicho autómata funciona. La intención es que los usuarios tengan un lugar donde experimentar con los autómatas así como poder compartirlos con otros usuarios.

Además de eso también se puedan crear y editar autómatas en la aplicación directamente de manera interactiva e intuitiva. Y se puedan buscar autómatas ya creados y publicados por otros usuarios así como compartir autómatas directamente con ellos.

1.2. Motivación

El Trabajo de Fin de Grado es la última asignatura de la carrera así que es necesario para un alumno el realizarlo. Este tiene como objetivo el poner a prueba los conocimientos del alumno mediante la aplicación de estos a un proyecto concreto, la realización de un informe y su presentación ante un tribunal para su evaluación.

La idea de realizar el TFG sobre la creación y compartición de autómatas surge tras realizar la asignatura de Gramática y Lenguajes Formales.

Dicha asignatura fue online debido al Coronavirus y durante ella, estudiando en grupo online, me di cuenta de lo complicado que era compartir autómatas. Por eso decidí hacer que mi TFG fuese el desarrollo de una aplicación que permitiese cumplir con esta necesidad que había encontrado.

1.3. Objetivos

El objetivo de este Trabajo de Fin de Grado es realizar una aplicación web en la que se implemente un simulador de autómatas (Autómatas Finitos Deterministas, Autómatas Finitos no Deterministas y Autómatas de Pila concretamente) con identificación de usuario que permita compartir autómatas entre usuarios, siempre y cuando estos estén añadidos (Sistema de amigos).

Los autómatas serán almacenados para que puedan ser accedidos posteriormente por el usuario que los guardó, bien sea para poder verlos, simularlos, compartirlos o ejecutarlos.

Además de esto, un objetivo fundamental será el aprendizaje y formación sobre la simulación de autómatas y el desarrollo de una aplicación web, tanto sobre su implementación como su teoría.

Capítulo 2

Metodología

Se ha empleado metodología Agile [1], el motivo de elegir esta sobre otras metodologías fue que el alumno ya tenía experiencia previa pues trabajó con ella durante las prácticas de empresa.

Se ha dividido el proyecto en 7 etapas de aproximadamente dos meses cada de una. La larga duración de las etapas ha sido a que al ser alumno de Indat al mismo tiempo que he estado desarrollando este TFG estaba desarrollando el TFG correspondiente a la parte de Estadística y al mismo tiempo finalizando las prácticas de empresa en las que había una probabilidad considerable de ser contratado a tiempo completo (cosa que sucedió).

Durante cada una de las fases se han realizado múltiples reuniones con el tutor para informar del estado actual del proyecto y recibir retroalimentación. El tiempo entre cada reunión ha sido habitualmente de dos semanas con alguna excepción debida a problemas de disponibilidad.

En la tabla 2.2 se puede ver una descripción general de la planificación inicial general. La duración planificada es contrastada con lo que ha durado de verdad en la tabla 2.3. Cada etapa se encuentra descrita con más detalle en su sub-apartado individual.

Etapa	Duración Estimada	Descripción Breve de los objetivos
Etapa 0: Planificación	22/2/2021-22/3/2021 (20h)	Planificación del alcance de la aplicación y definición de objetivos y casos de uso para etapas posteriores
Etapa 1: Entorno de Desarrollo	22/3/2021-22/5/2021 (30h)	Preparar el entorno de desarrollo así como búsqueda y recopilación de información de los recursos necesarios para su desarrollo
Etapa 2: Elaboración de un Prototipo	22/5/2021-22/7/2021 (60h)	Elaboración de un prototipo web implementado casos de uso relacionados con la cuenta de usuario (cumplir requisitos funcionales: RF1.X).
Etapa 3: Implementación de casos relacionados con usuarios	22/7/2021-22/9/2021 (60h)	Implementación los casos de uso relacionados con usuarios (cumplir requisitos funcionales: RF2.X).
Etapa 4: Elaboración del simulador de autómatas	22/9/2021-22/11/2021 (70h)	Escribir funciones y clases necesarias para poder modificar, simular y guardar autómatas (cumplir requisitos funcionales: RF3.X)
Etapa 5: Implementación del editor de autómatas	22/11/2021-22/2/2022 (50h)	Permitir al usuario visualizar y crear sus propios autómatas desde cero (cumplir requisitos funcionales: RF4.X)
Etapa 6: Implementación del diseño web definitivo	22/2/2022-22/3/2022 (40h)	Implementar el diseño web definitivo y realización de pruebas

Tabla 2.2: Planificación y descripción general2

Etapa	Duración Estimada	Duración Real	Diferencia
Etapa 0	20h	20h	0h
Etapa 1	30h	30h	0h
Etapa 2	60h	65h	+5h
Etapa 3	60h	60h	0h
Etapa 4	70h	77h	+7h
Etapa 5	50h	51h	+1h
Etapa 6	40h	41h	+1h
Total	330h	344h	+14h

Tabla 2.3: Duración Estimada vs Duración Real

2.1. Etapa 0

En esta etapa, a lo largo de múltiples reuniones, se ha planificado el resto del proyecto. Se ha determinado el alcance de la aplicación, sus requisitos y se ha determinado y descrito los casos de uso que deberán implementarse.

Una vez realizado lo anterior se ha estimado la carga de trabajo necesaria para completar las tareas y en base a ello, se han planificado las etapas subsiguientes del TFG teniendo en cuenta las condiciones del alumno (realización de otro TFG y prácticas de empresa en paralelo).

Tarea	Duración Estimada	Duración Real	Diferencia
Determinar alcance de la aplicación	1h	1h	0h
Determinar requisitos de la aplicación	4h	2h	-2h
Descripción de casos de uso	10h	12h	+2h
Estimación y planificación de etapas posteriores	5h	5h	0h
Total	20h	20h	0h

Tabla 2.4: Planificación etapa 0

2.2. Etapa 1

Esta etapa ha sido dedicada, a escoger y preparar el entorno de trabajo para que se puedan cumplir los requisitos planeados en la etapa anterior lo más sencillamente posible. Una parte importante de la preparación ha sido seleccionar y aprender las herramientas escogidas.

Tarea	Duración Estimada	Duración Real	Diferencia
Búsqueda de soluciones existente	5h	5h	0h
Búsqueda y Elección de Software	5h	5h	0h
Aprendizaje de OKTA [2]	2h	3h	+1h
Aprendizaje de D3 [3] (v3 [4])	2h	2h	0h
Aprendizaje de React [5]	2h	2h	0h
Aprendizaje de React-redux [6]	2h	2h	0h
Aprendizaje de Redux-saga [7]	2h	2h	0h
Aprendizaje de material-ui [8]	2h	1h	-1h
Aprendizaje de pg-promise [9]	2h	2h	0h
Preparación del entorno de desarrollo	6h	6h	0h
Total	30h	30h	0h

Tabla 2.5: Planificación etapa 1

2.3. Etapa 2

Un problema que ha surgido en esta etapa han sido los retrasos al realizar los casos de uso. Esto se ha debido a que ha sido bastante complicado encontrar la documentación en OKTA [2] para realizar tareas que se desvíen de la guía inicial usando su Widget [10]. A pesar de esto una vez encontradas la solución fue simple de implementar. Este retraso ha sido compensado en parte al Implementar el prototipo de la interfaz de usuario en menos tiempo del esperado.

Tarea	Duración Estimada	Duración Real	Diferencia
Diseño de la Interfaz de Usuario	20h	20h	0h
Implementar un Prototipo de la Interfaz	20h	15h	-5h
Implementar 'RF1.1 - Registrar Usuario'	4h	11h	+7h
Implementar 'RF1.2 - Iniciar Sesión'	4h	8h	+6h
Implementar 'RF1.3 - Restablecer Contraseña'	2h	1h	-1h
Implementar 'RF1.4 - Cerrar Sesión'	4h	1h	-3h
Implementar 'RF1.5 - Borrar Cuenta'	6h	9h	+3h
Total	60h	65h	+5h

Tabla 2.6: Planificación etapa 2

2.4. Etapa 3

Se procede expandiendo lo que es posible hacer en la aplicación, en este caso finalizando todos los casos de uso que incluyen a los usuarios.

Tarea	Duración Estimada	Duración Real	Diferencia
Implementar 'RF2.1 - Modificar Perfil de Usuario'	10h	10h	0h
Implementar 'RF2.2 - Buscar Amigo'	14h	14h	0h
Implementar 'RF2.3 - Añadir Amigos por Código Amigo'	8h	8h	0h
Implementar 'RF2.4 - Añadir Amigos por Búsqueda'	6h	6h	0h
Implementar 'RF2.5 - Eliminar Amigos'	10h	10h	0h
Implementar 'RF2.6 - Modificar Alias a Amigo'	6h	6h	0h
Implementar 'RF2.7 - Refrescar Lista Amigos'	6h	6h	0h
Total	60h	60h	0h

Tabla 2.7: Planificación etapa 3

2.5. Etapa 4

Una vez los casos de uso se han implementado en la etapa anterior se procede a implementar casos similares a los '2.X' pero para Autómatas.

La menor duración estimada en el desarrollo de los casos de uso RF3.1-RF3.8 (excluyendo RF3.3), se debe a que se ha asumido que su implementación serían muy parecida a la de sus correspondientes en 2.X.

Tarea	Duración Estimada	Duración Real	Diferencia
Implementar ’RF3.1 - Crear Autómata’	4h	4h	0h
Implementar ’RF3.2 - Eliminar Autómata’	4h	4h	0h
Implementar ’RF3.3 - Guardar Autómata’	8h	10h	+2h
Implementar ’RF3.4 - Buscar Autómatas’	4h	4h	0h
Implementar ’RF3.5 - Agregar Autómata Buscado’	4h	4h	0h
Implementar ’RF3.6 - Hacer público un Autómata’	4h	4h	0h
Implementar ’RF3.7 - Renombrar Autómata’	4h	4h	0h
Implementar ’RF3.8 - Recargar Lista Autómatas’	4h	4h	0h
Implementar ’RF3.9 - Compartir Autómata con Amigos’	12h	13h	+1h
Implementar ’RF3.10 - Validar Cadenas Entrada’	10h	12h	+2h
Implementar ’RF3.11 - Simular Autómata’	12h	14h	+2h
Total	70h	77h	+7h

Tabla 2.8: Planificación etapa 4

2.6. Etapa 5

En esta etapa se implementarán los últimos casos de uso que quedan. Además se añadirá un grafo a la aplicación para que dichos casos de uso puedan realizarse y comprobarse directamente desde la aplicación.

Tarea	Duración Estimada	Duración Real	Diferencia
Implementar ’RF4.1 - Añadir Estado’	10h	10h	Xh
Implementar ’RF4.2 - Borrar Estado’	10h	11h	+1h
Implementar ’RF4.3 - Añadir Transición’	10h	10h	Xh
Implementar ’RF4.4 - Borrar Transición’	10h	10h	Xh
Implementar ’RF4.5 - Editar Nombre Estado’	8h	11h	+1h
Implementar ’RF4.6 - Marcar o Desmarcar Estado como Final’	2h	1h	-1h
Total	50h	51h	0h

Tabla 2.9: Planificación etapa 5

2.7. Etapa 6

Etapa final del proyecto, se implementará la versión final de la interfaz de usuario basada en parte a la retroalimentación recibida por parte de los usuarios. También se realizará una corrección de los errores que se hayan ido dejando para el final debido a su baja prioridad.

Tarea	Duración Estimada	Duración Real	Diferencia
Modificación de la interfaz de usuario	20h	21h	+1h
Corrección de bugs dejados para el final	20h	20h	0h
Total	40h	41h	+1h

Tabla 2.10: Planificación etapa 6

2.8. Costes del Proyecto

Estimando que un ingeniero informático junior cobrará de media 13.28€ por hora[11], el coste en recursos puede estimarse en $13{,}28€/h * 330h = 4382{,}4€$ de manera inicial. El coste final siendo de $13{,}28 * 344 = 4568{,}32€$ una vez que se han añadido los retrasos al número de horas.

El material utilizado así como el coste que le ha supuesto al alumno se refleja en la siguiente tabla:

Material	Descripción	Coste
Portátil/Ordenador	Amortizado debido a su uso en proyectos anteriores	0€
Lugar de Trabajo	El alumno no paga por su lugar de residencia	0€
Software	Todo el software que a utilizar es gratuito o tiene licencia la universidad.	0€
Herramientas Físicas	Ya se disponen de folios de papel, bolígrafos, etc. de utilizarlos para estudiar la carrera	0€
Servidor	Máquina virtual proporcionada por la Universidad con el objetivo de realizar el TFG	0€
Internet y otros servicios	El alumno no paga por ellos	0€

Tabla 2.11: Coste de Materiales Utilizados

El coste estimado de este material sería de unos 610€ entre el Portátil (600€) y las Herramientas físicas (10€) pues es posible conseguir un Servidor gratuito para desarrollar la aplicación (aunque no para albergarla y esperar que estuviese operativa en producción) y es también posible utilizar la residencia actual como lugar de trabajo amortizando completamente su coste. Así mismo todo el Software utilizado es gratuito con la excepción de un repositorio en gitLab de la universidad que hubiera sido sustituido por un repositorio público en gitHub.

Parte II

Marco Conceptual y Contexto

Capítulo 3

Marco Conceptual

3.1. Conceptos Fundamentales

Antes de proseguir a las siguientes secciones es necesario definir unos conceptos básicos que aparecerán posteriormente. Estos son: Alfabeto, Cadena de Caracteres (o Input), Cadena Vacía y Longitud de una Cadena y Lenguaje, cuya definición en *Introducción a la teoría de autómatas, lenguajes y computación* [12] es:

- Alfabeto: Conjunto finito de Símbolos y no vacío. Se suele denotar por la letra $\sum$.
- Cadena de caracteres: Secuencia finita de símbolos seleccionados de algún alfabeto. El conjunto de todas las cadenas de caracteres de un alfabeto se denota por $\sum^*$.
- Cadena Vacía: Una cadena de caracteres con cero símbolos. Es denotada por la letra ϵ y su construcción es posible a partir de cualquier alfabeto.
- Longitud de una Cadena: Número de símbolos que tiene un cadena. La notación para indicar la longitud de una cadena w es $|w|$.
- Lenguaje: Conjunto de cadenas seleccionadas de un $\sum^*$ siendo $\sum$ un alfabeto determinado.

3.2. Autómatas

Un Autómata es una máquina abstracta de computo a la que dada una cadena de entrada ejecuta una secuencia de operaciones predeterminadas consumiendo un símbolo de la cadena por operación (incluida la cadena vacía dependiendo del tipo de autómata) hasta que la cadena de entrada sea consumida completamente, tras lo cuál determina si aceptar o rechazar la cadena.

Existen distintos tipos de Autómatas pero los que interesan para este Trabajo de fin de Grado son los Autómatas Finitos Deterministas (DFA), Autómatas Finitos no Deterministas (NFA) y Autómatas de Pila (PDA).

Los DFA y NFA forman parte de un grupo denominado Autómatas Finitos (FA). Éstos utilizan estados y transiciones entre estados en respuesta a las cadenas de entradas y son útiles para procesar texto, compiladores y diseño de hardware. La diferencia entre estos dos es el determinismo. Un DFA es determinista lo que significa que no puede estar en más de un estado al mismo tiempo, mientras que un NFA al ser no determinista pueden encontrarse en múltiple estados a la vez.

Un detalle importante sobre los DFA y NFA es que ambos son equivalentes (probado mediante construcción de subconjuntos [13]), es decir, es posible representar en ellos el mismo conjunto de lenguajes y es posible transformar cualquier DFA en un NFA equivalente y cualquier NFA en un DFA equivalente. Esto permite la construcción de NFAs para resolver problemas concretos y luego 'compilarlos' en un DFA para su ejecución.

Se puede dar la definición formal de un FA mediante la tupla $(Q, \sum, q_0, F, \delta)$:

- Q : Un conjunto finito de estados.
- $\sum$: Un alfabeto con los símbolos de entrada del autómata.
- q_0: Un estado inicial del autómata perteneciente a Q.
- F : Un subconjunto de Q que denota a los estados finales o aceptadores. Si cuando la cadena de entrada es consumida completamente el autómata se encuentra en uno de estos estados la cadena es aceptada, en caso contrario es rechazada.
- δ : Una función de transición, recibe un estado de entrada y un símbolo de $\sum$ y devuelve un subconjunto de estados de Q.

En el caso de los DFA el subconjunto de estados que devuelve la función de transición tiene un tamaño máximo de 1. Además de esto un DFA no acepta la cadena vacía como entrada a menos que el estado inicial pertenezca a los estados aceptadores.

Finalmente, un detalle importante de notar sobre los FA es que son fáciles de representar como grafos dirigidos, lo cual facilita su representación visual:

- Q : Es representado por el conjunto de nodos del grafo.
- q_0: El estado inicial del autómata. El autómata se encontrará en este estado antes de realizar ninguna transición. Esta transición es guardada fuera del grafo.
- F: Son, de nuevo, nodos del grafo. La información sobre que son aceptadores guardada fuera del grafo.
- δ: Son las aristas del grafo, teniendo en cada una los símbolos del alfabeto necesario para recorrerlas. El nodo inicial es el estado de entrada en la función de transición, el nombre de la arista es el símbolo de la función de transición y el nodo final el estado que devuelve la función. Un estado inicial conectado con múltiples nodos finales mediante aristas del mismo símbolo representarían una función de transición que devuelve múltiples estados. Las transiciones vacías se representan de la misma manera que el resto de transiciones pero utilizando como símbolo ϵ.
- $\sum$: Es el conjunto de distintos símbolos que aparecen en las aristas.

La única información que es necesario almacenar fuera del grafo es cual es el estado inicial y cuales son los estados aceptadores. Aunque pueden ser indicados

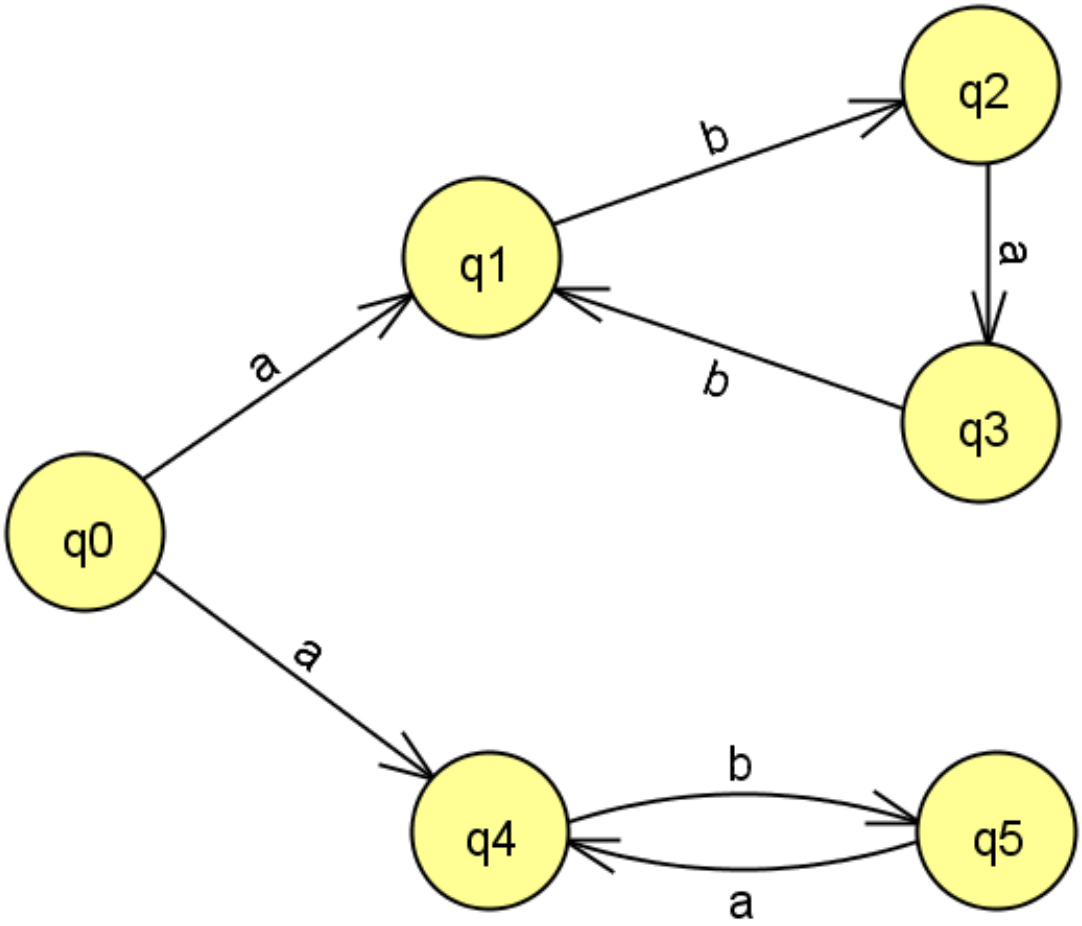

Figura 3.1: Representación en Grafo de un Autómata

El otro tipo de Autómatas que he mencionado son los PDA. Estos son una extensión de de los NFA con la adición de una Pila. Un PDA es capaz de representar lenguajes independientes de contexto, al contrario que los FA que solo son capaces de representar lenguajes regulares.

Una pila es un elemento en el que es posible almacenar una cadena de símbolos de pila, así como modificarlos. La información en una pila solo puede ser consultada de la misma forma que la de una pila LIFO (last-in First-out), es decir, solo es posible ver la 'cima', o primer símbolo introducido, de la pila.

Un PDA tiene dos formas distintas de aceptación, por estado final, el cual es el mismo que tienen los FA (Se acepta si cuando se termina de consumir la cadena de entrada el autómata se encuentra en un estado final) y por pila vacía (Se acepta si cuando se termina de consumir la cadena de entrada el autómata tiene la pila vacía). Ambos métodos de aceptación son equivalentes y es posible transformar un PDA por estado final a un PDA por pila vacía y viceversa.

Visto esto se puede dar la definición formal de un PDA mediante la tupla $(Q, \sum, q_0, \Gamma, Z_0, F, \delta)$:

- Q : Un conjunto finito de estados.
- $\sum$: Un alfabeto con los símbolos de entrada del autómata.
- q_0: El estado inicial del autómata. El autómata se encontrará en este estado antes de realizar ninguna transición
- Γ: Un alfabeto de pila finito. Indica el conjunto de símbolos que pueden introducirse en la pila
- Z_0: El símbolo inicial de la pila. Inicialmente la pila solo poseerá este símbolo y nada más.
- F : Un subconjunto de Q que denota a los estados finales o aceptadores.
- δ : La función de transición. Toma como argumentos de entrada un estado del autómata, un símbolo de $\sum$ y un símbolo de pila perteneciente a Γ. Esta función devuelve un conjunto finito de pares (p, Γ). p es el nuevo estado del autómata y Γ denota la cadena de símbolos por la que se va a reemplazar el símbolo en la cima de la pila. En el caso de que Y = ϵ se reemplaza por la cadena vacía , o lo que es lo mismo el símbolo en la cima de la pila se extrae. Por otro lado si Y = al símbolo en la cima de la pila se reemplaza por sigo mismo, es decir, este no cambia. Finalmente si Y es igual a cualquier otro cadena de símbolos se reemplaza el símbolo de la cima de la pila por estos, esto causa que la pila crezca.

Como se puede ver en la función de transición un PDA devuelve un conjunto de pares (p,Y), lo que implica que es posible que un PDA se pueda encontrarse en múltiples estados distintos y cada uno con un contenido distinto en la pila, o incluso en el mismo estado pero con múltiples contenidos de pila distintos. Lo cual a su vez complica su representación mediante un grafo.

3.3. Descripciones Instantáneas

Para poder simular los autómatas es necesario definir una forma de conocer el estado exacto en el que se encuentra un autómata en una simulación en un momento determinado. Esta definición es denominada descripción instantánea o configuración de un autómata y es diferente para cada uno de los distintos tipos de autómata descritos en la sección anterior. De acuerdo a [12] estas son:

- Para un DFA su descripción instantánea viene dada por la tupla (q, u) siendo q uno de los estados del autómata y u la cadena de entrada por consumir. Notese que $q \in Q$.
- Para un NFA su descripción instantánea consiste en un conjunto de tuplas (q, u) no repetidas. O lo que es lo mismo, un conjunto de configuraciones para un DFA.
- Para un PDA su descripción instantánea viene dada por un conjunto de tuplas (q, u, γ) con q siendo uno de los estados del autómata, u cadena de entrada por consumir y γ el conjunto de símbolos en la pila del autómata.

Las descripciones instantáneas permiten a partir de ellas generar otra descripción instantánea o aceptar o rechazar la cadena de entrada de entrada. También es posible almacenarlas en el caso de un simulación en el caso de que esta sea interrumpida para poder retomarla en otro momento.

3.4. Simulación

Un autómata recibe una cadena de entrada de $\sum^*$, siendo $\sum$ el alfabeto del autómata y la pregunta que se hace con el autómata es si la cadena de entrada introducida es aceptada o rechazada. Esto es equivalente a determinar si la cadena de entrada pertenece al alfabeto que describe el autómata. En este apartado se describen los pasos tomados para determinar esto.

3.4.1. DFA

La forma de generar una nueva configuración a partir de otra es simple para un DFA en comparación con otros autómatas debido a que no puede estar en múltiples estados a la vez, no tiene pila y su descripción instantánea solo consta de un única tupla (q, u) en lugar de un conjunto de tuplas.

1. Si $|u| \neq 0$ se coge el primer símbolo de u, al que denominaré a, y se ve si existe una transición desde el estado q a otro estado, que denominaré q_2 (notese que q_2 puede ser q), usando el símbolo a.

    a) Si $|u| = 0$ y $q \in F$ se acepta la cadena de entrada, en caso contrario se rechaza.

2. Si no existe ninguna transición desde q que consuma a se reemplaza q por q_2 y se elimina de u el símbolo en su primera posición, denominaré a esta nueva cadena de entrada u_2.

    a) Si no existe ninguna transición desde q que consuma a se rechaza la cadena de entrada

3. q_2 y u_2 forman la nueva descripción instantánea (u_2, q_2), con la que procedemos a realizar el mismo proceso hasta que se acepte o se rechaza la cadena de entrada.

3.4.2. NFA

Para simular un NFA se utilizará un algoritmo que simule también ϵ-NFA. La razón de esto es que un ϵ-NFA no es nada más que un NFA con transiciones-ϵ y por tanto un algoritmo que simule ϵ-NFA también será capaz de simular NFA.

La manera de simular un ϵ-NFA es distinta debido a que este autómata acepta cadenas vacías como entrada y su configuración instantánea está compuesta de un conjunto de tuplas (q, u). En un ϵ-NFA una tupla puede generar más de una tupla y puede tener transiciones-ϵ, y esto afecta a como se simula. La simulación se produce en tres paso: primero tratar con las transiciones-ϵ, segundo comprobar si $|u| = 0$ y tercero consumir el primer símbolo de u si la cadena de entrada no ha sido aceptada o rechazada. De estos tres pasos tanto el primero como el tercero pueden generar nuevas tuplas que son añadidas al conjunto.

1. Para cada tupla se generan todas las tuplas resultantes de transiciones-ϵ. De forma que, se obtendrán un conjunto de múltiples tuplas (q, u) con diferente q pero igual u, tal que: (q_0, u), (q_1, u), (q_2, u), etc. Estas tuplas son añadidas al conjunto inicial si no se encontraban en él.

a) Se repite el Paso 1 hasta que no se añada ninguna tupla al conjunto inicial debido a que todas las tuplas generadas son duplicadas.

2. Se crea un nuevo conjunto vacío, al que se irán añadiendo las nuevas tuplas que se generen y para cada tupla del conjunto al terminar el paso anterior se realiza el siguiente conjunto de pasos sucesivamente:

   a) Si $|u| \neq 0$ se consume el primer símbolo de u y se generan nuevas tuplas.

      1) Si $|u| = 0$ y $q \in F$ entonces se acepta la tupla y por tanto la cadena de entrada y la simulación termina. Si $q \notin F$ la tupla se marca como rechazada y se procede a la siguiente tupla.

   b) Para cada tupla de del conjunto generado en el Paso 2a se comprueba si existe ya en el nuevo conjunto mencionado en el Paso 2, si existe se descarta y si no se añade a él.

3. Si el conjunto del Paso 2 está vacío se rechaza la cadena de entrada, si no se vuelve al Paso 1 tomando como conjunto inicial las tuplas del Paso 2

3.4.3. PDA

Los PDA son simulados casi de la misma que un NFA, habiendo solo dos diferencias: el otro método de aceptación (pila vacía) y la pila, un elemento que no se encontraba en los NFA.

Tratar con el otro método de aceptación posible es trivial. Si el método de aceptación es 'estado final' entonces no hay que cambiar nada para tener en cuenta este cambio y si el método es 'pila vacía' entonces cualquier comprobación del tipo $q \in F$ pasa a ser $|\gamma| = 0$.

Esto es más complicado para la pila debido a que mientras que para un NFA el número máximo posible de tuplas en su descripción instantánea es de $(|u| + 1) * |Q|$, en un PDA la cantidad máxima de tuplas es infinita debido a que la pila puede almacenar una cantidad ilimitada de información. Esto provoca que un PDA no pueda simularse usando el mismo método que un NFA debido a que podría nunca pasarse del 'paso uno'.

Una forma fácil de ver esto es en un autómata con un estado con una transición hacía si mismo que consume la cadena vacía y añade un símbolo a la pila cada vez que se efectúa la transición.

Aquí el paso uno que sería efectuar todas las transiciones-ϵ, ya que las transiciones-ϵ en el estado mencionado 'nunca terminan' pues continúan generando tuplas nuevas no iguales (igual q e igual u pero con distinto γ).

Para tratar con este problema de forma adecuada es necesario hacer dos cambios:

El primero es eliminar el Paso 1a ('Se repite el Paso 1 hasta que no se añada ninguna tupla al conjunto inicial debido a que todas las tuplas generadas son duplicadas.'). De esta forma aunque hubiera un bucle como el que se ha explicado anteriormente con las transiciones-ϵ este no bloquearía los demás pasos.

El segundo es establecer un número máximo de veces que se ejecuta el algoritmo. Gracias al primer cambio el problema con las transiciones-ϵ no bloquea el algoritmo, pero si sucede sí evitará

que termine para un autómata cuya cadena de entrada no pertenezca al lenguaje que describe el autómata

En el caso de que la simulación se realice paso a paso, bastaría solo con aplicar el primer cambio ya que el usuario podría decidir manualmente sobre el segundo.

Una vez realizados estos cambios al algoritmo para los NFA es posible aplicar este directamente a los PDA, permitiendo su simulación.

Capítulo 4

Soluciones Existentes

La simulación de autómatas no es un tema nada nuevo, y esto significa que ya existan numerosas librerías, repositorios, páginas web y proyectos dedicados a este área de los lenguajes formales.

En este apartado voy a examinar algunas soluciones las cuales han servido al menos de referencia parcial para el diseño de la aplicación.

La primera es una página web[14], implementada en Javascript con Jquery[1] y jsplumb [2] que permite crear y simular autómatas de finitos (DFA y NFA) y de pila (PDA), aunque en este último caso no soporta la generación de descripciones instantáneas con diferentes pilas.

La simulación de autómatas puede ser la validación de una o múltiples cadenas de entradas o una simulación paso a paso (llamada 'mode debug'), donde se muestra cada paso que da el simulador.

Es también posible exportar (en local, texto plano o almacenamiento en web) e importar (de nuevo desde local, texto o web) los autómatas almacenados.

La siguiente es un repositorio de gitHub escrito por *calbeb531* [15], en Python que implementa una multitud de autómatas, entre ellos Autómatas Finitos y Autómatas de Pila, estos últimos con modo de aceptación tanto por pila vacía como por estado final.

Los autómatas se encuentran representados en formato JSON de forma que simplemente mirando el JSON es posible imaginar como será el autómata, o incluso dibujarlo sin muchas dificultades (cosa que no es posible por ejemplo en la aplicación web anterior)

Una cosa interesante a destacar, es que además también implementa distintas máquinas de Turing, tales como la máquina de Turing determinista, la no determinista y la multi-cinta no determinista. Si bien esto no es relevante para este trabajo al momento, podría ser una referencia útil en caso de que en un futuro se quiera expandir.

Otra solución existente es *PySimpleAutomata* [16]. Una librería escrita en Python 3 para administrar Autómatas Finitos y Autómatas de alternación Finito de Palabras (AFW).

[1] https://jquery.com/
[2] https://github.com/jsplumb/jsplumb

Utiliza DOT, un lenguaje de descripción de grafos en texto plano para representar gráficamente los autómatas finitos, aunque estos también pueden ser representados en formato JSON. Por otro lado los AFW solo pueden ser representados en JSON ya que estos no tienen una representación natural en forma de grafo.

La manera de representar los autómatas finitos en formato JSON es distinta del repositorio de *caleb531*, siendo menos legible para una persona y más complicada de trabajar con ella a la hora de programar.

Otra herramienta es JFLAP [17], JFLAP es un Software de código libre escrito en Java para experimentar con lenguajes formales y distintos tipos de autómatas, entre otras cosas. Entre estos autómatas se incluyen DFA, NFA y PDA. JFLAP es con claridad, de entra las soluciones vistas, la solución que más opciones ofrece en cuanto a lenguajes formales. Además de esto es también la que mejor las implementa en términos de corrección de la solución. Debido a estar hecho en Java los autómatas son objetos en java y no se representan usando JSON.

Finalmente también está el repositorio *Converting-a-NFA-to-DFA* de *SachinKumar105* [18]. El cual implementa formas de transformar un Autómata Finito no Determinista a determinista. La forma en que los autómatas deben de ser representados para que puedan ser transformados es además muy similar a como deben representarse para la solución del repositorio de *calbeb531*.

Parte III

Desarrollo del Sistema

Capítulo 5

Análisis

5.1. Descripción del Modelo de Usuario

Debido a las características y diseño de la aplicación es necesario que las personas que quieran usarla posean un conocimiento básico de los distintos tipos de autómatas y como funcionan, esto es debido a que para editar y simular autómatas es necesario saber que es un autómata, que elementos tiene, etc. Por tanto los usuario son personas que utilizan la aplicación y poseen un conocimiento básico de teoría de autómatas.

Teoría de autómatas es impartida en cursos universitarios y por tanto se estima que la mayoría de usuarios sean o estudiantes o profesores universitarios. Teniendo esto en cuenta se estima que la edad de estas usuarios estará entre 18 (edad de entrada a la universidad, que es donde se imparte teoría de autómatas) y 67 (edad de jubilación) años y su género será irrelevante.

Si bien es posible que haya personas que la usen fuera del rango de edad [18-67], bien sea porque estas personas son autodidactas y están aprendiendo por su cuenta, o bien porque son profesores impartiendo que han decidido no jubilarse a los 67 años o bien por otros motivos, se espera que este número sea muy bajo en comparación a los que se encuentren en dicho rango y por tanto no se tendrán en consideración.

En base a este rango de edad sobre los usuarios, su nivel de estudios, y sus estudios, es razonable asumir que poseen conocimientos básicos de como usar el ordenador, internet, registrarse en una página web y confirmar su registro mediante correo electrónico.

Estas cuatro asunciones son fundamentales pues la aplicación es una aplicación web y es necesario registrarse y confirmar correo para poder utilizarla. Si bien puede utilizarse parcialmente sin registrarse es necesario estar registrado para poder utilizar toda la funcionalidad de la aplicación

5.2. Requisitos Funcionales

Teniendo en cuenta la planificación se han agrupado los requisitos funcionales en cuatro grupos.

1. RF1: Requisitos de gestión de sesión de usuario: Incluye todos los requisitos que están relacionados con la cuenta y sesión del usuario que usa la aplicación. Son todos los requisitos que requieren de comunicación con OKTA.
2. RF2: Requisitos de administración de usuarios: Incluye todos los requisitos relacionados con los usuarios de la aplicación excepto los que tienen que ver con su cuenta o sesión. Todos estos requisitos requieren de comunicación con el servidor de la aplicación, bien sea para actualizar la base de datos o para obtener información de ella.
3. RF3: Requisitos de autómatas como unidad abstracta: Incluye los requisitos relacionados con la manipulación de un autómata a nivel de archivo o elemento. Todos estos requisitos, excepto el RF3.10 y RF3.11 requieren de comunicación con el servidor de la aplicación, bien sea para actualizar la base de datos o para obtener información de ella.
4. RF4: Requisitos de manipulación del autómata: Incluye todos los requisitos relacionados con operaciones sobre el autómata mismo. Todos estos requisitos son realizados completamente en el lado del cliente de la aplicación, sin comunicación con el servidor.

RF1: Requisitos de Gestión de Sesión de Usuario

ID	Nombre del Requisito	Descripción del Requisito
RF1.1	Registrar Usuario	El sistema debe permitir a un usuario registrarse en el sistema.
RF1.2	Iniciar Sesión	El sistema debe permitir que un usuario inicie sesión
RF1.3	Restablecer Contraseña	El sistema debe tener un sistema que permita al usuario restablecer su contraseña en caso de que se le haya olvidado su contraseña
RF1.4	Cerrar Sesión	El sistema debe permitir a un usuario identificado cierre sesión
RF1.5	Borrar Cuenta	El sistema debe permitir a un usuario identificado borrar su cuenta de usuario permanentemente

Tabla 5.2: Requisitos de Gestión de Sesión del Usuario

RF2: Requisitos de Administración de Usuarios

ID	Nombre del Requisito	Descripción del Requisito
RF2.1	Modificar Perfil de Usuario	El sistema debe permitir a un usuario identificado cambiar su nombre de usuario, generar un nuevo código de amigo y cambiar sus opciones de configuración (entre ellas la de si es un usuario público o no)
RF2.2	Buscar Amigo	EL sistema debe permitir a un usuario buscar a usuarios públicos
RF2.3	Añadir Amigos por Código Amigo	El sistema debe permitir a un usuario identificado agregar a otros usuarios identificados como amigos
RF2.4	Añadir Amigos por Búsqueda	El sistema debe permitir a un usuario identificado buscar y agregar a otros usuarios públicos
RF2.5	Eliminar Amigos	El sistema debe permitir a un usuario identificado eliminar a un amigo agregado
RF2.6	Modificar Alias a Amigo	EL sistema debe permitir a un usuario identificado modificar el alias de un amigo
RF2.7	Refrescar Lista de Amigos	El sistema debe permitir a un usuario identificado volver a cargar todos los amigos de su lista.

Tabla 5.4: Requisitos de Administración de Usuarios

RF3: Requisitos de Autómatas como Unidad Abstracta

ID	Nombre del Requisito	Descripción del Requisito
RF3.1	Crear Autómata	El sistema debe permitir a un usuario identificado crear un autómata finito o de pila
RF3.2	Eliminar Autómata	El sistema debe permitir a un usuario identificado eliminar un autómata propio guardado permanentemente
RF3.3	Guardar Autómata	El sistema debe permitir a un usuario identificado almacenar un autómata para su uso posterior
RF3.4	Buscar Autómatas	EL sistema debe permitir a un usuario buscar autómatas públicos
RF3.5	Agregar Autómata Buscado	EL sistema debe permitir a un usuario agregar a sus autómatas una copia de un autómata público buscado
RF3.6	Hacer público un Autómata	El sistema debe permitir a un usuario identificado renombrar un autómata propio
RF3.7	Renombrar Autómata	El sistema debe permitir a un usuario identificado renombrar un autómata propio
RF3.8	Recargar Lista de Autómatas	El sistema debe permitir a un usuario reemplazar su lista actual de autómatas con los que se encuentran almacenados en la base de datos.
RF3.9	Compartir Autómata con Amigos	EL sistema debe permitir a un usuario identificado compartir un autómata con un amigo
RF3.10	Validar Cadenas de Entrada	El sistema debe permitir a un usuario simular un autómata
RF3.11	Simular Autómata	El sistema debe permitir a un usuario simular un Autómata Paso a Paso, mostrando al usuario las descripciones instantáneas del autómata tras cada paso.

Tabla 5.6: Requisitos de Autómatas como Unidad Abstracta

RF4: Requisitos de Manipulación del Autómata

ID	Nombre del Requisito	Descripción del Requisito
RF4.1	Añadir Estado	El sistema debe permitir a un usuario identificado crear un nuevo estado en un autómata
RF4.2	Borrar Estado	El sistema debe permitir a un usuario identificado borrar un estado existente en un autómata
RF4.3	Añadir Transición	El sistema debe permitir a un usuario identificado crear una nueva transición de un estado existente a otro estado existente en un autómata
RF4.4	Borrar Transición	El sistema debe permitir a un usuario identificado borrar una transición existente en un autómata
RF4.5	Editar Nombre de Estado	El sistema debe permitir a un usuario identificado renombrar un estado del autómata con un nombre ya asignado.
RF4.6	Marcar o Desmarcar Estado como Final	El sistema debe permitir a un usuario identificado marcar o desmarcar un estado como estado final.

Tabla 5.8: Requisitos de Manipulación del Autómata

De la misma manera también se han agrupado los diagramas como las descripciones de los casos de uso

5.3. Diagramas de Casos de Uso

Se identifican a dos actores: Usuarios y Usuarios Identificado. Un Usuario es cualquier Usuario que utilice la aplicación. Mientras que un Usuario Identificado es un Usuario que tiene iniciada sesión en una cuenta en la aplicación.

Requisitos de Gestión de Sesión de Usuario

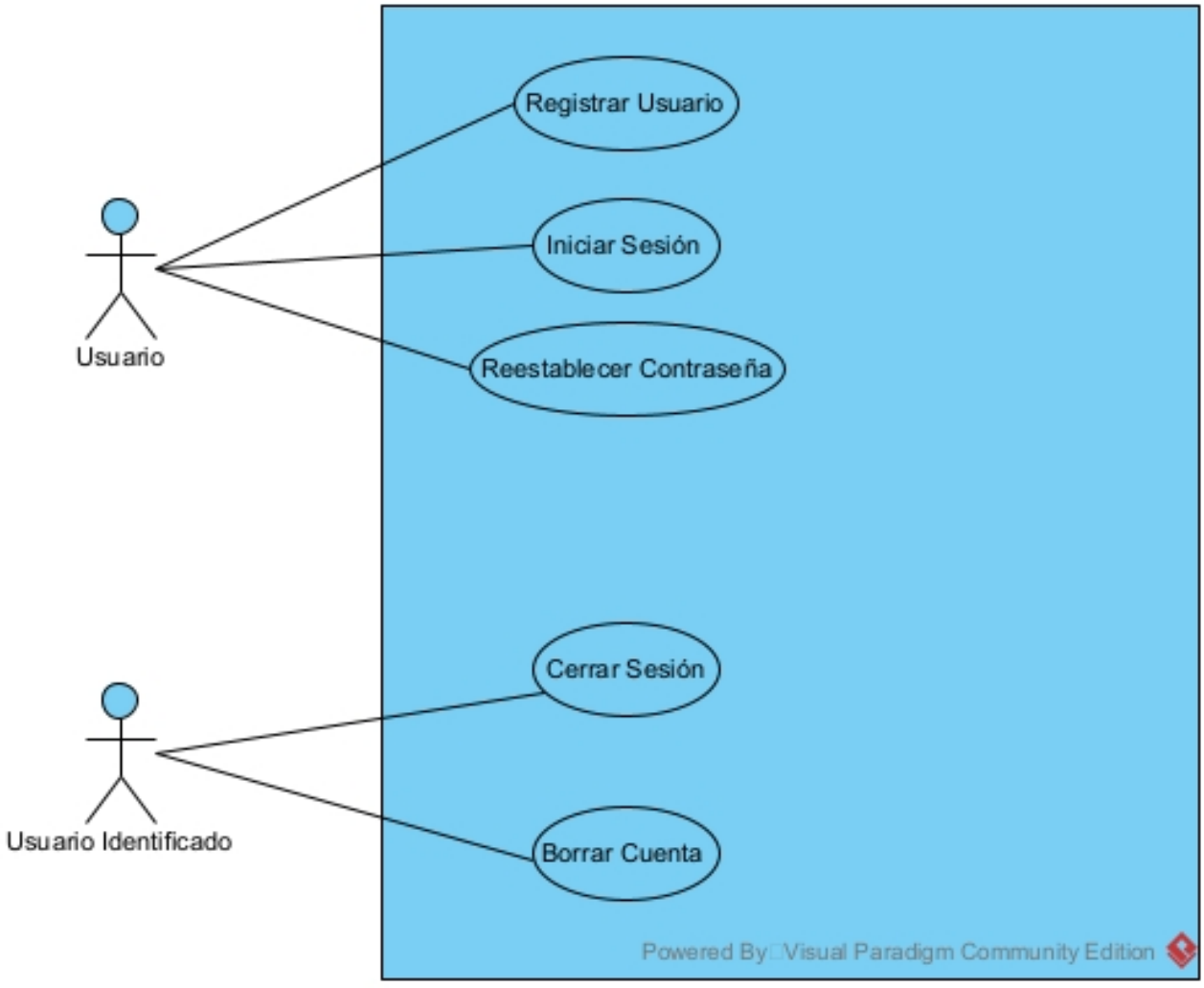

Figura 5.1: Diagrama de Casos de Uso de Gestión de Sesión del Usuario

RF1.1 - Registrar Usuario	
Actor	Usuario
Precondición	El usuario trata de registrarse
Postcondición	El usuario a registrarse se encuentra registrado en el sistema, y con su sesión iniciada.
Secuencia Normal	1. El sistema pide al usuario los siguientes datos: Email, Contraseña y Nombre de Usuario (en la aplicación) 2. El usuario introduce los datos pedidos y los continua. 3. El sistema envía la solicitud de registro a OKTA con los datos, el cuál los verifica y agrega al usuario a su base de datos de la aplicación y envía un email de confirmación al usuario. 4. El sistema pide al usuario que confirme su email y le ofrece volver a la página de inicio de sesión. 5. El usuario confirma su email y es redirigido a la página de la aplicación (con su sesión no iniciada)
Alternativas y Excepciones	1a. Si el usuario vuelve al inicio de Sesión el caso de uso queda sin efecto y se inicia al caso de uso 'RF 1.2 - Iniciar Sesión' 2a. Si el usuario ya se encuentra registrado el sistema informa al usuario de esto y vuelve al punto 1 pero conservando los datos que el usuario había introducido 2b. Si el usuario introduce una contraseña invalida el sistema informa al usuario de esto y vuelve al punto 1 pero conservando los datos que el usuario había introducido, excepto la contraseña 3a. Si Okta determina que los datos del usuario son incorrectos o su contraseña no es segura se salta la paso 1

Tabla 5.10: RF1.1 - Registrar Usuario

RF1.2 - Iniciar Sesión	
Actor	Usuario
Precondición	El usuario tiene una cuenta creada en el sistema
Postcondición	El usuario tiene su sesión iniciada.
Secuencia Normal	1. El Usuario introduce su email y contraseña y trata de iniciar sesión. También, de manera opcional, puede indicar al Sistema que lo recuerde para las siguientes veces. 2. El Sistema verifica la información introducida con la que hay en el servidor de Okta e inicia la Sesión del usuario, permitiéndole acceder a su cuenta. 3. El Sistema manda un request (con el id de Okta del usuario y su username) al servidor pidiéndole los datos de usuario, sus autómatas y su lista de contactos. 4. El Servidor obtiene de la base de datos los datos pedidos y se los devuelve al Sistema. 5. El Sistema recibe los datos y los carga en la aplicación
Alternativas y Excepciones	1a. Si el Sistema recuerda al Usuario entonces se salta directamente al paso 2. Sin necesidad de input del Usuario, usando el email y contraseña que el Sistema recuerda. 1b. Si el usuario trata de restablecer su contraseña este caso de uso queda sin efecto y se inicia el Caso de Uso 'RF 1.3 - Restablecer Contraseña' 2a. Si el Usuario no se encuentra registrado en el sistema, o su contraseña o nombre de usuario es incorrecto, el sistema informará de ello al usuario y vuelve al paso 1. 2b. Si el Usuario no ha confirmado su email, informará de que no se puede iniciar sesión y se vuelve al paso 1. 2c. Si el Usuario inicia Sesión correctamente indicando al Sistema que lo recuerde para veces posteriores, entonces las siguientes veces que acceda a la página web su sesión estará iniciada. 4a. Si el sistema no al usuario en la base de datos entonces crea una nueva entrada para el usuario usando como id el id de Okta que ha revibido, username el username que ha recibido y el resto valores por defecto

Tabla 5.12: RF1.2 - Iniciar Sesión

RF1.3 - Restablecer Contraseña	
Actor	Usuario
Precondición	El usuario tiene una cuenta ya creada en el sistema
Postcondición	La contraseña del Usuario es cambiada
Secuencia Normal	1. El Usuario introduce su email y trata de reestablecer su contraseña mediante correo electrónico 2. El Sistema envia un email al Usuario con un enlace a su email. 3. El Usuario va a su email y hace click en el enlace enviado. 4. El Enlace rediríge al Usuario a una página web donde le pide al Usuario que introduzca su nueva contraseña y que la confirme. 5. El Usuario introduce su nueva contraseña dos veces y confirma. 6. El Sistema cambia la contraseña del Usuario y lo redirige a su Página de usuario en Okta.
Alternativas y Excepciones	1a. Si el Usuario cancela el caso de uso queda sin efecto y termina. 3a. Si el Usuario cancela el caso de uso queda sin efecto y termina. 4a. Si el Enlace ha expirado el Usuario es redirigido a una página donde se le informa de esto y el Caso de Uso termina y queda sin efecto. 4b. Si hay algún error el Usuario es redirigido a una página donde se le informa del error y el Caso de Uso termina y queda sin efecto. 5a. Si el Usuario cancela el caso de uso queda sin efecto y termina. 6a. Si la Contraseña y la Confirmación de la Contraseña no coinciden se informa de esto al Usuario y se vuelve al paso 5. 6b. Si la Contraseña es invalida se informa de esto al Usuario y se vuelve al paso 5.

Tabla 5.14: RF1.3 - Restablecer Contraseña

RF1.4 - Cerrar Sesión	
Actor	Usuario Identificado
Precondición	Ninguna
Postcondición	El usuario ya no tiene una sesión iniciada en el sistema
Secuencia Normal	1. El Usuario trata de cerrar sesión. 2. El Sistema notifica a Okta de que la sesión del usuario a finalizado y lo redirige a la página principal
Alternativas y Excepciones	3a. Si el usuario no trata de cerrar sesión el Caso de Uso es termina y queda sin efecto.

Tabla 5.16: RF1.4 - Cerrar Sesión

RF1.5 - Borrar Cuenta	
Actor	Usuario Identificado
Precondición	Ninguna.
Postcondición	La sesión del Usuario es cerrada y su cuenta borrada
Secuencia Normal	1. El Usuario trata de borrar su cuenta de usuario 2. El Sistema le pide confirmación al usuario informándole de que esta acción es irreversible 3. El Usuario confirma 4. El Sistema envía un request Okta para que desactive la cuenta de usuario 5. El Sistema envía un request a Okta para que borre la cuenta del usuario 6. El Sistema borra al usuario de la base de datos así como todos los autómatas de los que es dueño y todos los contactos relacionados con él.
Alternativas y Excepciones	3a. El Usuario cancela, y el Caso de Uso termina

Tabla 5.18: RF1.5 - Borrar Cuenta

RF2: Requisitos de Administración de Usuarios

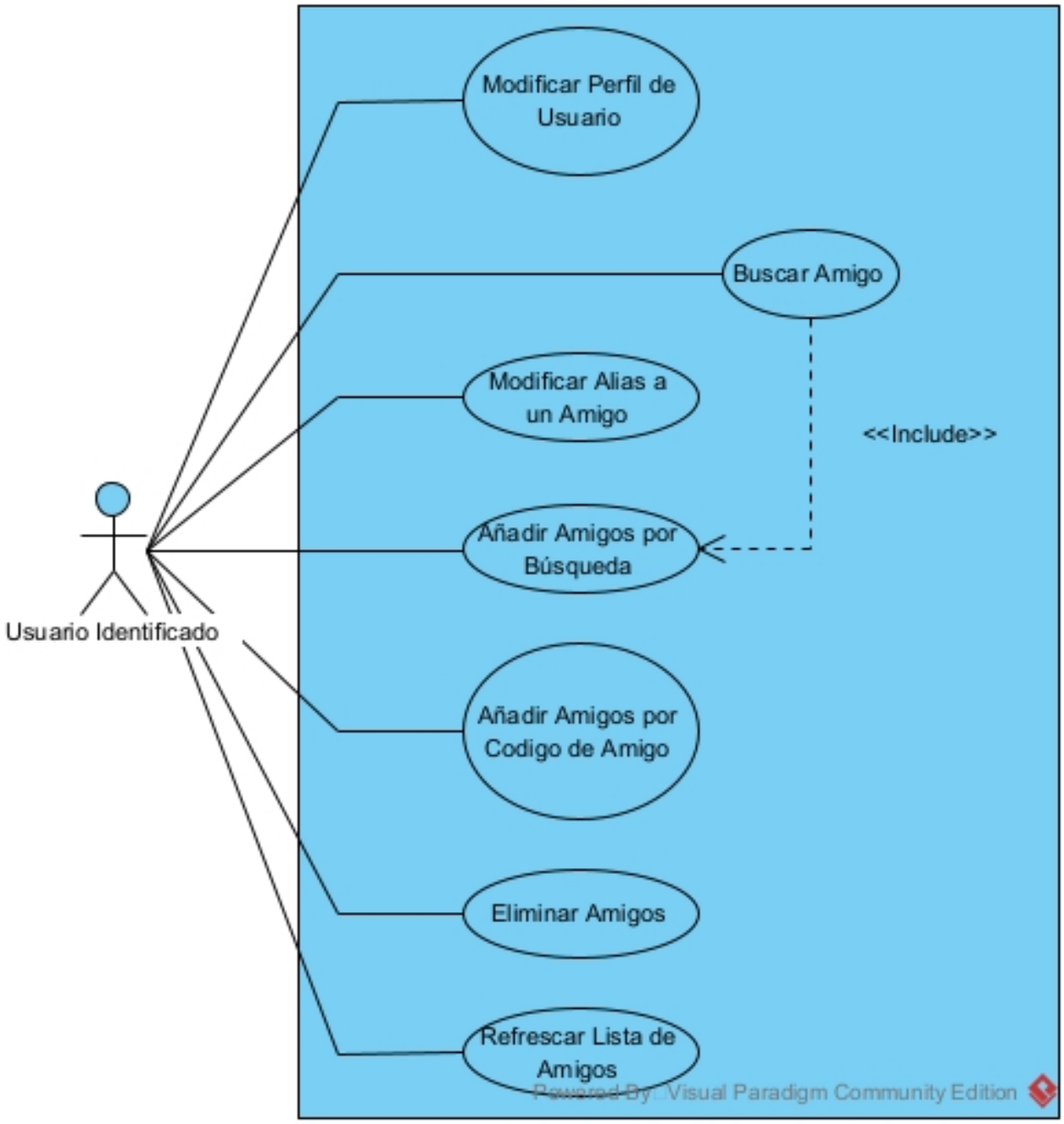

Figura 5.2: Diagrama de Casos de Uso de Administración de Usuario

RF2.1 - Modificar Nombre de Usuario	
Actor	Usuario Identificado
Precondición	Ninguna.
Postcondición	Ninguna.
Secuencia Normal	1. El Usuario edita su perfil de Usuario: editando su nombre directamente, modificando sus opciones de configuración o generando un nuevo código de amigo 2. El Usuario guarda los cambios 3. El Sistema almacena las nuevas opciones de configuración, nombre de usuario y código de amigo 4. El Sistema actualiza el nombre de usuario del usuario con el que ha recibido (si no ha sido modificado entonces el nombre no cambiará) 5. El Sistema actualiza el código de amigo del usuario con el que ha recibido (si no se ha generado uno nuevo entonces el código de amigo no cambiará) 6. El Sistema actualiza las opciones de configuración del usuario con las que ha recibido (si alguna de ellas no ha cambiado entonces esa no cambiará)
Alternativas y Excepciones	2a. Si el usuario no guarda los cambios entonces el caso de uso termina y queda sin efecto.

Tabla 5.20: RF2.1 - Modificar Nombre de Usuario

RF2.2 - Buscar Amigo	
Actor	Usuario Identificado
Precondición	Ninguna.
Postcondición	Ninguna.
Secuencia Normal	1. El Usuario introduce una cadena de caracteres e indica si esta buscando usuarios públicos o usuarios en su lista de contactos. 2. El Sistema busca todos los usuarios cuyo nombre de usuario incluye exactamente la cadena de caracteres introducida 3. El Sistema filtra los usuarios de acuerdo a si son públicos o están en la lista de contactos del Usuario. 4. El Sistema devuelve el resultado de la búsqueda y de los filtros al usuario.
Alternativas y Excepciones	

Tabla 5.22: RF2.2 - Buscar Amigo

RF2.3 - Añadir Amigo por Código Amigo	
Actor	Usuario Identificado
Precondición	El Usuario conoce el código de amigo del Usuario que quiere agregar.
Postcondición	Ninguna
Secuencia Normal	1. El Usuario intenta agregar un amigo por código de amigo. 2. El Sistema le pide que introduzca el código de amigo del usuario a agregar y un alias para el usuario. 3. El Usuario introduce identificador, un alias para el amigo a agregar y confirma. 4. El Sistema busca al usuario y si lo encuentra nuevas entrada en la lista de amigo de ambos usuarios con estados y mensajes adecuados. Ambas entradas también se guardan en la base de datos
Alternativas y Excepciones	1a. Si el Usuario no intenta agregar un amigo por código de amigo el Caso de Uso queda sin efecto y termina. 3a. Si el Usuario cancela el envío de la solicitud entonces Caso de Uso queda sin efecto y termina. 3.b Si el Usuario no introduce ningún alias el sistema usar el Username del usuario a agregar en ese momento. 3.c Si el Usuario introdujese un código de amigo con formato erróneo el Sistema alerta de esto al usuario y no deja intentar agregar amigo hasta que este correcto 4.a Si el sistema no encuentra al usuario entonces el caso de uso termina y queda sin efecto.

Tabla 5.24: RF2.3 - Añadir Amigo por Código Amigo

RF2.4 - Añadir Amigo por Búsqueda	
Actor	Usuario Identificado
Precondición	El Usuario ha realizado una búsqueda de usuarios públicos o de autómatas públicos.
Postcondición	Ninguna
Secuencia Normal	1. El Usuario trata de agregar uno de los usuarios que puede agregar 2. El Sistema busca al usuario y si lo encuentra crea una nueva entrada en la lista de amigo de ambos con estados mensajes adecuados. Ambas entradas son almacenadas en la base de datos
Alternativas y Excepciones	2a. Si el Sistema no encuentra al Usuario el caso de uso queda sin efecto y termina.

Tabla 5.26: RF2.4 - Añadir Amigo por Búsqueda

RF2.5 - Eliminar Amigo	
Actor	Usuario Identificado
Precondición	El Usuario tiene al menos un contacto.
Postcondición	Ninguna.
Secuencia Normal	1. El Usuario trata de Eliminar un Contacto. 2. El Sistema pide confirmación al usuario. 3. El Usuario confirma. 4. El Sistema elimina el amigo de la lista de contactos del usuario y al usuario de la lista de contactos del amigo así como las entradas en la base de datos
Alternativas y Excepciones	2a. Si el usuario tiene desmarcada como opción en sus ajustes el preguntar por confirmación antes de borrar amigos entonces se salta al Paso 4. 3a. Si el Usuario cancela el caso de uso queda sin efecto y termina.

Tabla 5.28: RF2.5 - Eliminar Amigo

RF2.6 - Modificar Alias de Amigo	
Actor	Usuario Identificado
Precondición	El Usuario tiene al menos un Amigo.
Postcondición	Ninguna.
Secuencia Normal	1. El Usuario trata de cambiar el alias de un contacto. 2. El Sistema le muestra el alias actual del contacto y le permite editarla. 3. El Usuario edita dicha alias en la que quiera y confirma. 4. El Sistema cambia el alias del usuario y guarda los cambios
Alternativas y Excepciones	1a. Si el Usuario no trata de cambiar el alias de un contacto el Caso de Uso queda sin efecto y termina. 3a. Si el Usuario cancela la introducción de un alias entonces Caso de Uso queda sin efecto y termina. 3b. Si el Usuario deje el alias en blanco entonces el Sistema no le deja confirmar, informando al Usuario de que no puede dejar el nuevo alias en blanco.

Tabla 5.30: RF2.6 - Modificar Alias de Amigo

RF2.7 - Refrescar Lista de Amigos	
Actor	Usuario Identificado
Precondición	Ninguna..
Postcondición	Ninguna.
Secuencia Normal	1. El Usuario trata de recargar la lista de amigos. 2. El Sistema obtiene todos los amigos que el usuario posee en el momento actual. 3. El Sistema reemplaza todos los amigos en la lista de amigos con todos los que ha obtenido.
Alternativas y Excepciones	

Tabla 5.32: RF2.7 - Refrescar Lista de Amigos

RF3: Requisitos de Autómatas como Unidad Abstracta

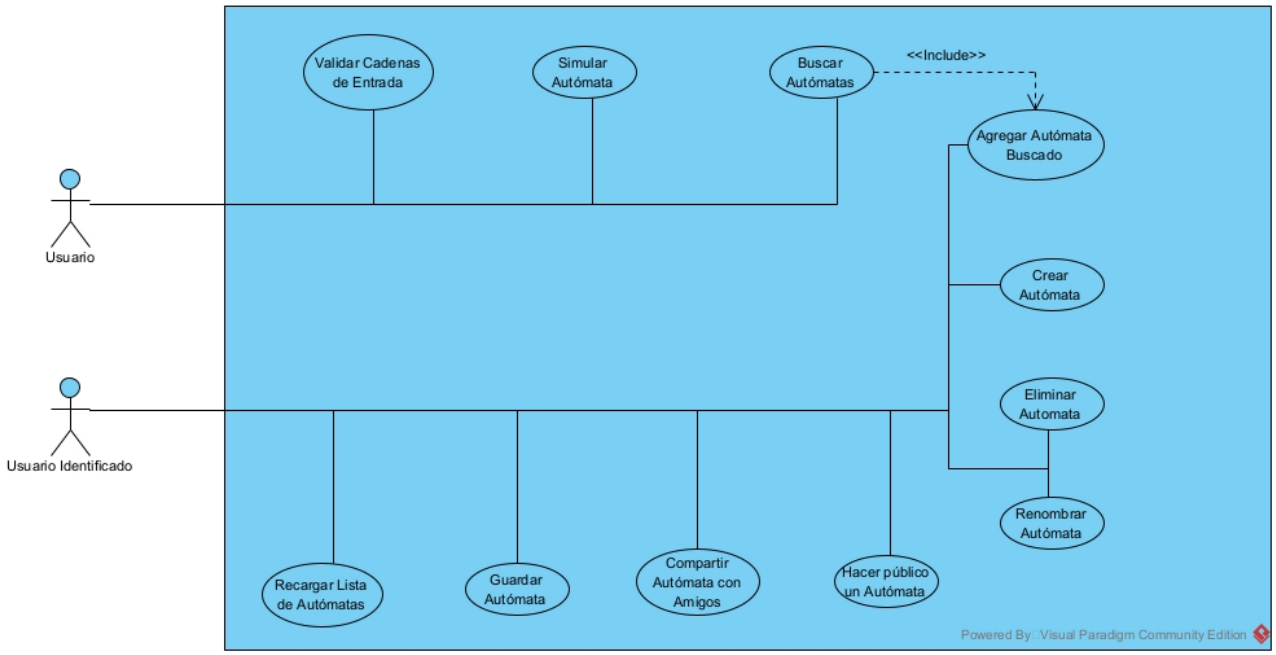

Figura 5.3: Diagrama de Casos de Uso de Autómatas como Unidad Abstracta

RF3.1 - Crear Autómata	
Actor	Usuario Identificado
Precondición	Ninguna.
Postcondición	Ninguna
Secuencia Normal	1. El Usuario trata de crear un autómata. 2. El Sistema pregunta al Usuario el tipo (DFA, NFA, PDA por pila vacía y PDA por estado final) y nombre del Autómata a crear. 3. El Usuario introduce el nombre y tipo de Autómata. 4. El Sistema verifica los datos introducidos. 5. El Sistema crea una entrada para el autómata en la base de datos y añade al autómata a la lista de autómatas del usuario
Alternativas y Excepciones	3a. Si el Usuario cancela la creación del autómata el caso de uso termina y queda sin efecto. 3.1b. Si el Usuario selecciona como tipo de Autómata un PDA', el Sistema le pide al usuario que introduzca los símbolos iniciales en la Pila. 3.2b. El Usuario selecciona el método de aceptación y los símbolos iniciales en la pila y se salta al Paso 4. 4a. Si hay algún campo vacío el Sistema informa de esto al Usuario y se salta al Paso 3.

Tabla 5.34: RF3.1 - Crear Autómata

RF3.2 - Eliminar Autómata	
Actor	Usuario Identificado
Precondición	El usuario tiene al menos un autómata
Postcondición	Ninguna
Secuencia Normal	1. El Usuario trata de eliminar un autómata. 2. El Sistema pide confirmación al usuario. 3. El Usuario confirma que quiere eliminar el autómata. 4. El Sistema elimina el autómata
Alternativas y Excepciones	2a. Si el Usuario tiene en sus ajustes de configuración que no se pida confirmación al borrar un autómata se salta al paso 4. 3. Si el usuario cancela el caso de uso termina y queda sin efecto.

Tabla 5.36: RF3.2 - Eliminar Autómata

RF3.3 - Guardar Autómata	
Actor	Usuario Identificado
Precondición	Hay un Autómata en el Editor de Autómatas
Postcondición	Ninguna
Secuencia Normal	1. El Usuario selecciona guardar autómata. 2. El Sistema guarda los cambios en el Autómata y manda un request al servidor para que actualice el Autómata con los cambios realizados en la base de datos.
Alternativas y Excepciones	

Tabla 5.38: RF3.3 - Guardar Autómata

RF3.4 - Buscar Autómatas	
Actor	Usuario o Usuario Identificado
Precondición	Ninguna.
Postcondición	Ninguna.
Secuencia Normal	1. El Usuario introduce una cadena de caracteres y, opcionalmente, filtros para el tipo de autómata que esta buscando. 2. El Sistema busca todos los autómatas cuyo nombre de usuario incluye exactamente la cadena de caracteres introducida 3. El Sistema filtra los autómatas de acuerdo a los filtros (si hay al menos un filtro que sea 'Incluir' solo incluirá autómatas que se correspondan con esos tipo, si solo hay filtros 'Excluir' excluirá autómatas que se correspondan con esos tipo. Si no hay filtros incluirá todos) 4. El Sistema devuelve el resultado de la búsqueda.
Alternativas y Excepciones	

Tabla 5.40: RF3.4 - Buscar Autómata

RF3.5 - Añadir Autómatas por Búsqueda	
Actor	Usuario Identificado
Precondición	El Usuario ha realizado una búsqueda de autómatas públicos.
Postcondición	Ninguna
Secuencia Normal	1. El Usuario trata de agregar uno de los autómatas 2. El Sistema crea una copia del autómata en la base de datos con el user_id igual al del usuario que lo esta tratando de agregar
Alternativas y Excepciones	

Tabla 5.42: RF3.5 - Añadir Autómatas por Búsqueda

RF3.6 - Hacer Público el Autómata	
Actor	Usuario Identificado
Precondición	El usuario tiene al menos un autómata
Postcondición	Ninguna
Secuencia Normal	1. El Usuario trata de hacer público el autómata. 2. El Sistema cambia el estado del autómata a público si antes no lo era o viceversa y actualiza su estado en la base de datos.
Alternativas y Excepciones	

Tabla 5.44: RF3.6 - Hacer Público el Autómata

RF3.7 - Renombrar Autómata	
Actor	Usuario Identificado
Precondición	Hay un autómata en la lista de autómatas
Postcondición	Ninguna
Secuencia Normal	1. El Usuario trata de renombrar un autómata. 2. El Sistema muestra el nombre actual del autómata y le permite editarlo. 3. El Usuario edita el nombre del Autómata y confirma. 4. El Sistema renombra el Autómata y manda un request al servidor para que se guarda el renombramiento en la base de datos.
Alternativas y Excepciones	1a. Si el Usuario hace click fuera de la lista de Autómatas y de la lista de acciones mostrada el Caso de Uso queda sin efecto y termina. 3a. Si el Usuario cancela el Caso de Uso queda sin efecto y termina.

Tabla 5.46: RF3.7 - Renombrar Autómata

RF3.8 - Recargar Lista de Autómatas	
Actor	Usuario Identificado
Precondición	Ninguna.
Postcondición	Ninguna.
Secuencia Normal	1. El Usuario trata de recargar la lista de autómatas. 2. El Sistema obtiene todos los autómatas que el usuario posee en el momento actual. 3. El Sistema reemplaza todos los amigos en la lista de autómatas con todos los que ha obtenido.
Alternativas y Excepciones	

Tabla 5.48: RF3.8 - Recargar Lista de Autómatas

RF3.9 - Compartir Autómata con Amigos	
Actor	Usuario Identificado
Precondición	El usuario tiene al menos un autómata y al menos un amigo en estado aceptado.
Postcondición	Ninguna.
Secuencia Normal	1. El Usuario trata de Compartir un Autómata. 2. El Sistema muestra al Usuario su lista de amigos aceptados. 3. El Usuario selecciona tantos amigos como quiera de la lista y confirma el compartir el autómata. 4. El Sistema envía el autómata a todos los amigos seleccionados acompañado por un mensaje indicando quien compartió el autómata con ellos
Alternativas y Excepciones	3a. Si el Usuario cancela entonces el Caso de Uso queda sin efecto y termina. 3b. Si el Usuario no selecciona al menos 1 amigo entonces el Sistema no deja al usuario compartir el autómata.

Tabla 5.50: RF3.9 - Compartir Autómata con Amigos

RF3.10 - Validar Cadenas de Entrada	
Actor	Usuario o Usuario Identificado
Precondición	Hay un Autómata en el Simulador de Autómatas
Postcondición	Ninguna
Secuencia Normal	1. El Usuario introduce tantas cadenas de entrada como quiera simular, separándolas por un salto de línea. 2. El Usuario trata de validar dichas cadenas de entrada. 3. El Sistema toma las cadenas de entrada y simula cada una. 4. El Sistema muestra al usuario los resultados de la simulación.
Alternativas y Excepciones	1a. Si el Usuario no ha introducido ninguna cadena de entrada se considerará que ha introducido una cadena vacía. 3a. Si el Autómata que se está simulando es PDA y se producen más de 10000 pasos en la simulación de un input sin que este sea aceptado o rechazado el Sistema rechazará arbitrariamente esa cadena de caracteres

Tabla 5.52: RF3.10 - Validar Cadenas de Entrada

RF3.11 - Simular Autómata	
Actor	Usuario o Usuario Identificado
Precondición	Hay un Autómata en el Simulador de Autómatas
Postcondición	Ninguna
Secuencia Normal	1. El Usuario introduce una única cadena de caracteres a simular y trata de simularla. 2. El Sistema crea una descripción instantánea inicial con toda la cadena de caracteres introducida como input y estado inicial 'start'. 3. El Sistema ofrece al usuario la opción 'continuar' o de 'detener' la simulación. 4. El Usuario decide si 'continuar' o 'detener'. 5. Si el Usuario continua el Sistema genera y muestra nuevas descripciones instantáneas basadas en las descripciones instantáneas inmediatamente anteriores.
Alternativas y Excepciones	1a. Si el Usuario introduce un salto de línea se introducirá un espacio en blanco en su lugar. 4a. Si el Usuario cancela Caso de Uso queda sin efecto y termina, pero las descripciones instantáneas generadas siguen mostrándose. 5a. Si alguna descripción instantánea es aceptada se vuelve imposible 'continuar' y se salta al paso 4. 5b. Si todas las descripciones instantáneas son rechazadas se vuelve imposible 'continuar' y se salta al paso 4.

Tabla 5.54: RF3.11 - Simular Autómata

RF4: Requisitos de Manipulación del Autómata

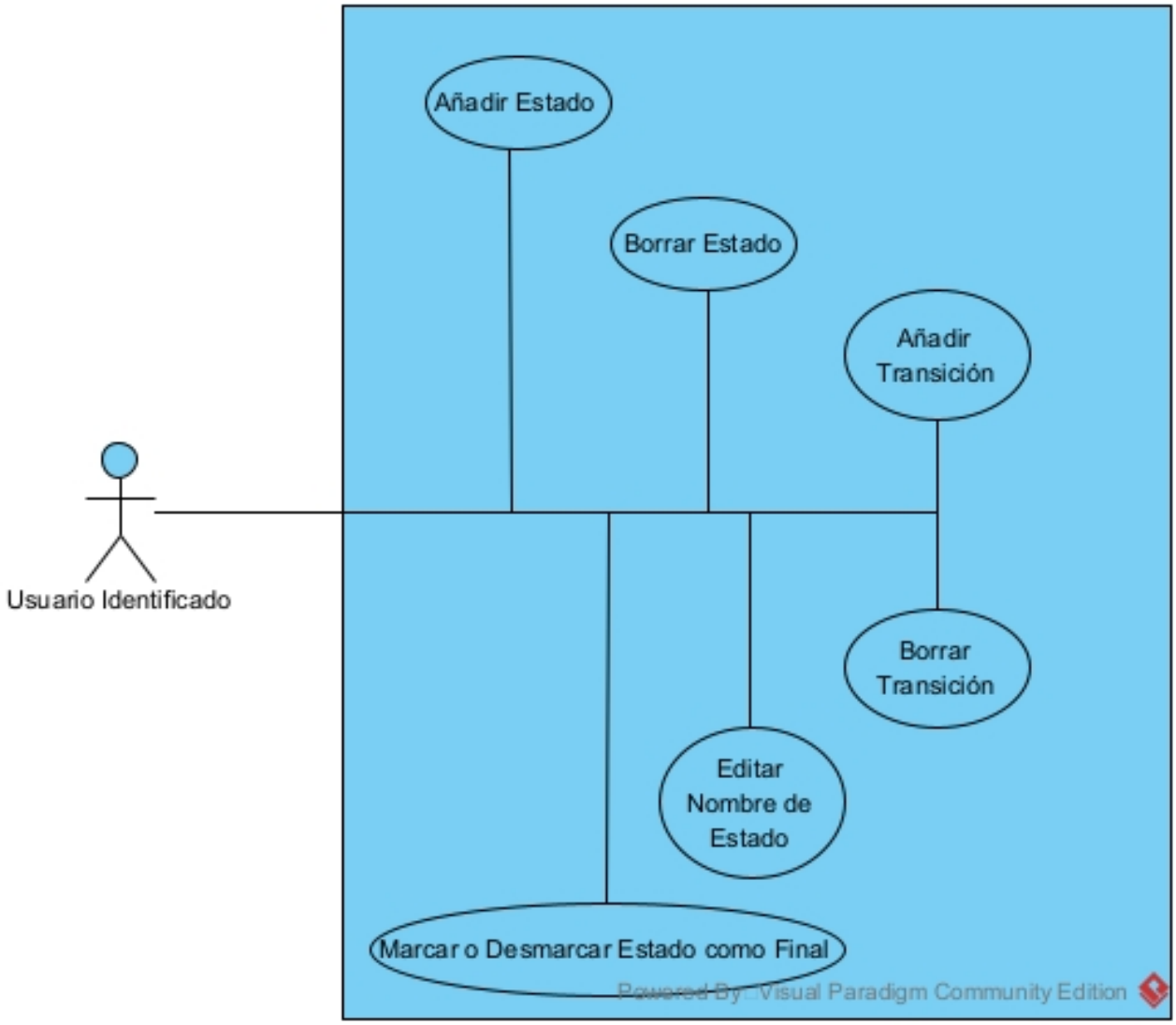

Figura 5.4: Diagrama de Casos de Uso de Manipulación del Autómata

RF4.1 - Añadir Estado	
Actor	Usuario Identificado
Precondición	Hay un Autómata en el Editor de Autómatas y el usuario trata de añadir un estado a dicho autómata
Postcondición	Ninguna
Secuencia Normal	1. El Sistema crea un nuevo estado con un nuevo id (invisible para el usuario) y un nombre asignado automáticamente y deja al usuario editar el nombre. 3. El Usuario edita el nombre y confirma. 4. El Sistema cambia el nombre para el estado del autómata. 5. El Sistema incrementa el valor del atributo 'idct' del autómata en el editor en 1.
Alternativas y Excepciones	3a. Si el usuario no edita el nombre entonces se salta al paso 4.

Tabla 5.56: RF4.1 - Añadir Estado

RF4.2 - Borrar Estado	
Actor	Usuario Identificado
Precondición	Hay un Autómata en el Editor de Autómatas con al menos un estado seleccionado excepto el estado 'start' y el usuario trata de borrarlo
Postcondición	Ninguna
Secuencia Normal	1. El Sistema elimina todas las transiciones/edges asociados al estado (que tengan a dicho estado como estado de llegada o como estado de salida) a borrar 2. El Sistema elimina el estado del autómata y el nodo del autómata
Alternativas y Excepciones	1a. Si el estado a borrar es el estado 'start' entonces el caso de uso queda sin efecto y termina

Tabla 5.58: RF4.2 - Borrar Estado

RF4.3 - Añadir Transición	
Actor	Usuario Identificado
Precondición	Hay un Autómata en el Editor de Autómatas con al menos un estado y el trata de crear una transición entre 2 estados
Postcondición	Ninguna
Secuencia Normal	1. El Sistema pregunta al Usuario por el símbolo a consumir si es un autómata finito y por el símbolo, símbolo en la cima de la pila y símbolos por los que reemplazar la cima de la pila si es un PDA. 2. El Usuario introduce la información pedida. 3. El Sistema verifica la información que introduce el usuario es correcta para el tipo de autómata. 4. El Usuario confirma que desea crear la transición. 5. El Sistema crear la transición en el autómata.
Alternativas y Excepciones	3a. La verificación será incorrecta para un autómata DFA: si el símbolo es el carácter vacío, si ya existe una transición desde un 'estado A' a un 'estado B' con el mismo símbolo a consumir o si ya existe la misma transición (estado salida, símbolo a consumir, estado llegada) en el autómata. 3b. La verificación será incorrecta para un autómata NFA: si ya existe la misma transición (estado salida, símbolo a consumir, estado llegada) en el autómata. 3c. La verificación será incorrecta para un autómata PDA: si se deja la cima de la pila vacía o si la transición ya existe en el autómata.

Tabla 5.60: RF4.3 - Añadir Transición

RF4.4 - Borrar Transición	
Actor	Usuario Identificado
Precondición	Hay un Autómata en el Editor de Autómatas con al menos una transición
Postcondición	Ninguna
Secuencia Normal	1. El Usuario selecciona la transición a borrar y trata de borrarla. 2. El Sistema elimina la transición del autómata. 3. El Sistema borra la transición que el usuario había hecho click.
Alternativas y Excepciones	1a. Si el Usuario no selecciona una transición y trata de borrarla entonces el Caso de Uso queda sin efecto y termina.

Tabla 5.62: RF4.4 - Borrar Transición

RF4.5 - Editar Nombre de Estado	
Actor	Usuario Identificado
Precondición	Hay un Autómata en el Editor de Autómatas con al menos un estado
Postcondición	Ninguna
Secuencia Normal	1. El Usuario trata de renombrar un estado. 2. El Sistema hace editable el texto del nodo del estado que trata de editar 3. El Usuario edita el texto del nodo a su gusto 4. El Sistema cambia el nombre del nodo al texto editado.
Alternativas y Excepciones	3a. Si el Usuario no edita el texto del nodo el caso de uso termina y queda sin efecto.

Tabla 5.64: RF4.5 - Editar Nombre de Estado

RF4.6 - Marcar o Desmarcar Estado como Final	
Actor	Usuario Identificado
Precondición	Hay un Autómata en el Editor de Autómatas con al menos un estado y el usuario trata de marcar un estado como final
Postcondición	Ninguna
Secuencia Normal	1. El Sistema pone el estado como final si y lo añade a la lista de estados finales, en caso de que ya lo fuera lo elimina de la lista de estados finales y lo desmarca
Alternativas y Excepciones	1a. Si el estado era un estado final entonces el Sistema lo quita de estado final y lo elimina de la lista de estados finales.

Tabla 5.66: RF4.6 - Marcar o Desmarcar Estado como Final

5.4. Descripción del Modelo de Dominio

El modelo de dominio tiene el objetivo de representar el vocabulario y los conceptos clave del dominio del problema. El modelo de dominio también identifica las relaciones entre todas las entidades comprendidas en el ámbito del dominio del problema, y comúnmente identifica sus atributos. [19]

Por convenio en UML se utiliza un diagrama de clases para representar el dominio. El siguiente diagrama de clases describe los objetos que son necesarios para el desarrollo de la aplicación.

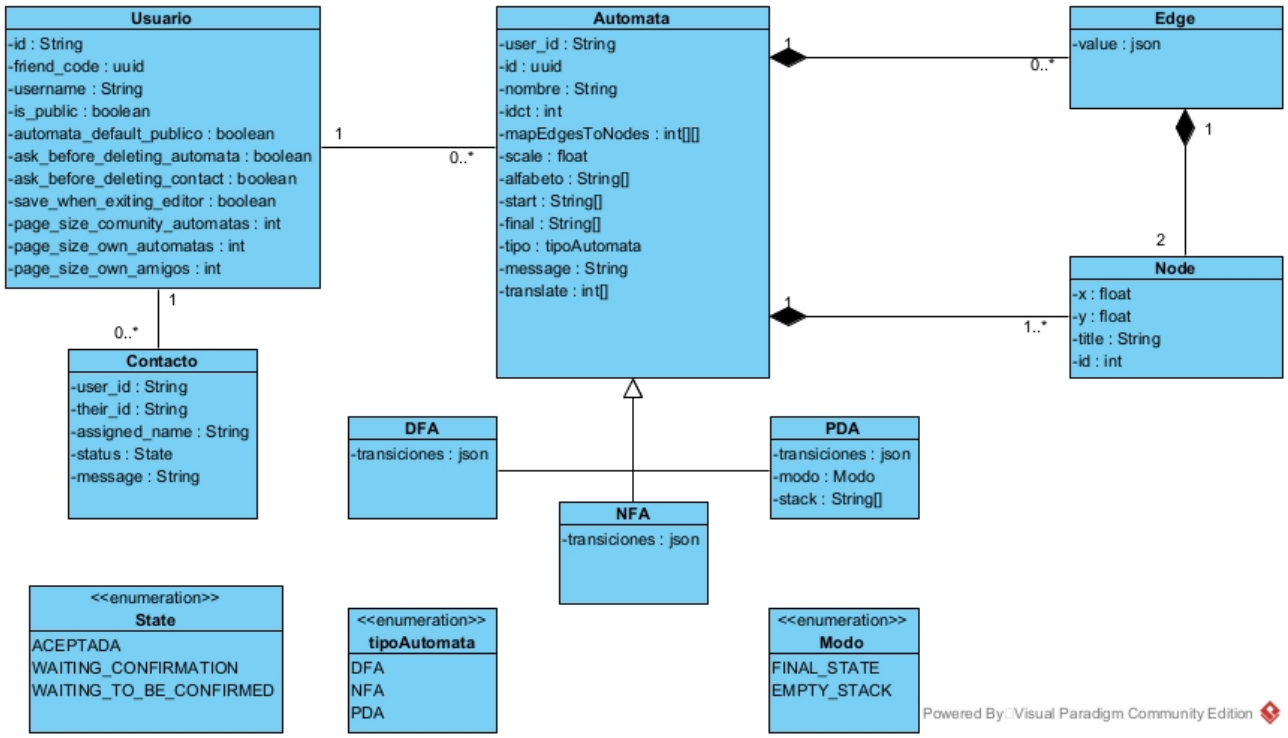

Figura 5.5: Diagrama del Modelo de Dominio

A continuación se procede a describir las clases y sus relaciones detalle en mayor detalle.

Usuario: representa a un usuario que haya iniciado sesión en la aplicación. Un usuario que no haya iniciado sesión también podrá utilizar ciertas funciones de esta pero no es necesario modelarlo pues solo estará usando autómatas ya creados por Usuarios registrados los cuales no podrá modificar.

- id: El id asignado por OKTA a un usuario para la aplicación dada. OKTA garantiza su unicidad [20].
- friend_code: código de amigo de un usuario. Se utilizará para poder agregar a un usuario mediante él.
- username: nombre que un usuario se asigna a si mismo. Se utilizará para poder buscar a otros usuarios.
- is_public: campo que determina si un usuario puede ser buscado por otros usuarios.
- resto de campos: distintas opciones de configuración del usuario.

Contacto: representa una relación entre dos Usuarios. A través de este objeto es posible determinar si dos Usuarios son amigos y enviarse autómatas. Siempre habrá un número par de Contactos uno correspondiente a 'Usuario A siendo contacto de Usuario B' y otro a 'Usuario B siendo contacto de Usuario A'.

- user_id: el id correspondiente al usuario que posee el contacto.
- their_id: el id correspondiente al usuario del que se ha hecho el contacto.
- assigned_name: alias que se ha otorgado al usuario.
- status: estado del amigo. No se considera un estado REJECTED porque en caso de que esto suceda simplemente se borrara el amigo.
    1. WAITING_CONFIRMATION: se ha preguntado a tratado de agregar a otro usuario y se esta a la espera de que este confirme o rechace. Equivalente a recibir una solicitud de amistad y esperar por confirmación.
    2. WAITING_TO_BE_CONFIRMED: un usuario ha tratado de agregarle y esta esperando a que usted rechace o acepte.
    3. ACCEPTED: se ha preguntado o recibido una solicitud y se ha confirmado.
- message: cadena de caracteres con objetivo de informar al usuario de su estado. Ejemplo: 'El usuario Adrián Sebastián ha pedido ser su amigo'.

Node: representa un Nodo del grafo que representará a un Autómata visualmente. Al mismo tiempo cada objeto Node del grafo se corresponderá estrictamente con un estado del autómata. Por simplicidad y para evitar errores un autómata siempre tendrá al menos un estado (y por tanto un Node) que será el estado inicial.

- x: coordenada x del centro del nodo para ser representado.
- y: coordenada y del centro del nodo para ser representado.
- title: nombre del nodo que se mostrará al usuario.
- id: identificador del nodo. Debe ser único para un autómata

Edge: representa una arista dirigida de un nodo a otro en un grafo. Esta compuesta por 2 1 nodo 'source' denotando el inicio de la transición y un nodo 'target' denotando el final. Es posible que el nodo 'source' y el nodo 'target' sean el mismo.

- value: Indica el tipo de autómata como key al que el objeto pertenece y como valor los valores de la transición (un único carácter para los DFA, un array de caracteres para los NFA y un array de arrays de Strings para los PDA).

Autómata: Objeto formado por múltiples Node (mínimo 1 pues se ha considerado que siempre se creará con un nodo, 'start', que será imborrable para evitar problemas potenciales) y cero o más Edge (pues es posible construir un autómata con un único nodo).

- user_id: id del usuario que es dueño del autómata
- id: identificador único del autómata.
- nombre: nombre asignado por un usuario al autómata.
- idct: campo que determina que id asignar a los nuevos Node del autómata. Empieza en 2.
- mapEdgesToNodes: es necesario que los Edge referencien directamente a los Node (no vale solo con su valor), este campo existe para realizar dicha asignación eficientemente en caso de que un autómata sea traído desde la base de datos.
- scale: valor que determina el zoom del grafo.
- alfabeto: contiene todos los caracteres que consume un autómata.
- start: contiene el id del estado inicial para empezar a simular. Esta incluido por completitud pues dicho valor no cambia y siempre es el nodo 'start'
- final: contiene los id de los estados finales. Usado para determinar si una cadena es aceptada o rechazada (excepto para los PDA por pila vacía).
- tipo: tipo del autómata. DFA, NFA o PDA.
- translate: array de dos valores que se corresponderán a los del atributo translate en un svg. Determinan cuanto transladar los Node del grafo.
- message: cadena de caracteres utilizada si se desea informar de algo al usuario como: 'El usuario Adrián Sebastián a compartido este autómata con usted'.

Además de esto se puede observar la existencia de 3 clases hijo que heredan de Autómata, DFA, NFA y PDA. Estas tres clases poseen el mismo atributo y con el mismo tipo ('transiciones : json'). Cada uno de dichos jsons tiene un formato distinto que representa adecuadamente las transiciones de dicho autómata.

Esto se deba a que no es lo mismo la transición de un DFA en donde se consume un símbolo para llegar a un estado que la de un PDA donde se consume un símbolo, se cambia el estado de la pila y se pueden llegar a múltiples estados.

Los PDA además tienen como atributos adicionales 'Modo' y 'Stack'. Modo determina el método de aceptación, estado final o pila vacía, mientras que Stack indica el conjunto inicial de símbolos en la Pila.

Asociaciones: Entre las clases descritas anteriormente existen múltiples asociaciones, estas denotan el propósito de una clase de interactuar con otro, estas son las siguientes:

- Usuario-Contacto: un Usuario tiene 0 o más Contactos, pero para que un Contacto exista es necesario que exista un Usuario.
- Usuario-Autómata: un Usuario tiene 0 o más Autómatas, pero para que un Autómata exista es necesario que exista un Usuario.
- Autómata-Node: un Autómata está compuesto de 1 (nodo inicial que no se puede borrar, hecho así por decisiones de diseño) o más Nodes. un Node representa un estado del autómata.

- Autómata-Edge: un Autómata esta compuesto por 0 (autómata con estado inicial igual a estado final sin ninguna transición y que acepte la cadena vacía) o más Edge. Un Edge representa una transición entre estados del autómata.
- Edge-Node: un Edge está compuesto por dos Nodes, un Node de salida y uno de llegada.

Capítulo 6

Diseño

6.1. Patrones de Diseño Aplicados

Los patrones de diseño son unas técnicas para resolver problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces.

Un patrón de diseño resulta ser una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser reutilizable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias [21].

React utiliza de forma nativa el patrón observador mientras que React-redux utiliza el patrón Singleton.

El patrón Observador es un patrón utilizado para notificar a unos observadores que están suscritos a un objeto de he este ha cambiado.

Por otro lado el Patrón Singleton es utilizado para garantizar que a lo largo de la aplicación exista una única instancia de un objeto, y de proveer una forma de acceder a dicho objeto.

React usa el patrón observador para saber cuando tiene que volver a renderizar un componente porque ha habido cambios en su estado, mientras que por otro lado Singleton es utilizado por react-redux para garantizar que solo haya un único estado global en la aplicación.

También se ha utilizado el patrón fachada para tratar con los autómatas. Debido a que hay 3 tipos distintos de clase de Autómatas cuando hay que realizar una operación sobre ellos se llama a la fachada la cual determina que tipo de autómata es y que argumentos necesita el método que se esta llamado y ejecuta la función correspondiente.

Además de esto también se ha utilizado la arquitectura VMC (Vista-Modelo-Controlador) con Base de Datos.

El propósito del patrón VMC es la separación de los datos y su lógica de como se representan y

del módulo que se encarga de gestionar las comunicaciones entre los datos y su representación.

Como se representan los datos es la vista, los datos y su lógica es el modelo y el módulo que gestión la comunicación entre ambos es el controlador. En la aplicación la Base de Datos en conjunto con el estado global de la aplicación actúan son el modelo.

6.2. Diseño de la Representación Interna de los Autómatas

Para garantizar que un autómata pueda ser representado de manera correcta y simulado posteriormente su implementación interna debe poder ser mapeada a cada uno de los elementos de un autómata explicados en el marco teórico.

Para los autómatas finitos esto se traduce en

- Q : El conjunto de caracteres, representado en el atributo 'transiciones'.
- $\sum$: El alfabeto, representado en el atributo 'alfabeto'. Un array de Strings de un único carácter sin repetir.
- q_0: El estado inicial, representado por el atributo 'start'. Un array de tamaño uno con el id del estado inicial.
- F : El conjunto de estados aceptadores, representados por el atributo 'final'. Un array con los ids de los estados aceptadores
- δ : La función de transición, representada en el atributo 'transiciones'

Mientras que los autómatas de pila tienen como partes adicionales:

- Γ: En alfabeto de pila finito. No se representa pues no es necesario ya que el usuario solo puede añadir un número finito de caracteres a la pila.
- Z_0: El símbolo inicial de la pila, representado por Stack. Un array de Strings de un único carácter.

De estas partes, alfabeto, estado inicial y estados finales son fáciles, sin embargo, esto es algo más engañoso con 'transiciones' ya que este es un JSON con estructura distinta para cada tipo de autómata.

Podría haberse empleado un array de arrays y en cada celda poner los caracteres a consumir cuando se realiza la transición, sin embargo, esto hubiera sido poco recomendable para grandes autómatas debido al crecimiento $O(n^2)$ en espacio con el número de estados. Además de que este método se vuelve incluso peor en términos de espacio para los PDA al tener que acomodar para la existencia de la pila.

Un JSON permite evitar este problema ya que en lugar de almacenar todas las transiciones posibles solo almacena aquellas que existan actualmente en el autómata. Con este propósito se ha usado la siguiente estructura para el JSON de los autómatas finitos:

$id\_estado\_salida- > simbolo\_a\_consumir- > [id\_estado\_llegada\_1, id\_estado\_llegada\_2, ...]$

Sin embargo, esta estructura no puede emplearse para los autómatas de pila debido a la existencia de la pila. Para este caso se ha usado la siguiente estructura:

$id\_estado\_salida- > simbolo\_a\_consumir- > smbolo\_cima\_pila- > id\_estado\_llegada- > [[reemplazamiento\_cima\_pila1,1,..], [reemplazamiento\_cima\_pila2,1,..], ...]$

Por conveniencia se explicará primero cada termino de las estructuras excepto el último que se explicará aparte:

- id_estado_salida: El id del Node del estado de salida.
- $smbolo\_a\_consumir$: Un único carácter que se eliminará de la cadena de entrada si se realiza la transición
- $id\_estado\_llegada\_1$: El id del Node del estado de llegada.
- $smbolo\_cima\_pila$: El símbolo que debe estar en la cima de la pila para que la transición ocurra.
- $reemplazamiento\_cima\_pila1,1$: Símbolo por el que se va a reemplazar la cima de la pila. Es un String de un único carácter.

Ahora sobre los últimos términos de las estructuras. Para el caso de los autómatas finitos es un array con múltiples ids de para los estados de llegada. Esto es así para acomodar el hecho de que en los NFA es posible pasar de un único estado a múltiples al consumir un símbolo, en caso de que sea un DFA este array tendrá tamaño 1 y contendrá un único id.

En el caso de los PDA es un array de un array de Strings. Empezando por el final, los arrays con múltiples $reemplazamiento\_cima\_pila1$ están hay porque es posible reemplazar la cima de la pila por más o menos de un único símbolo Símbolo, mientras que el hecho de que sea un array de arrays es porque es posible que un PDA no solo se encuentre en diferentes estados en un mismo momento si no que su pila también sea distinta.

6.3. Diseño de la Base de Datos

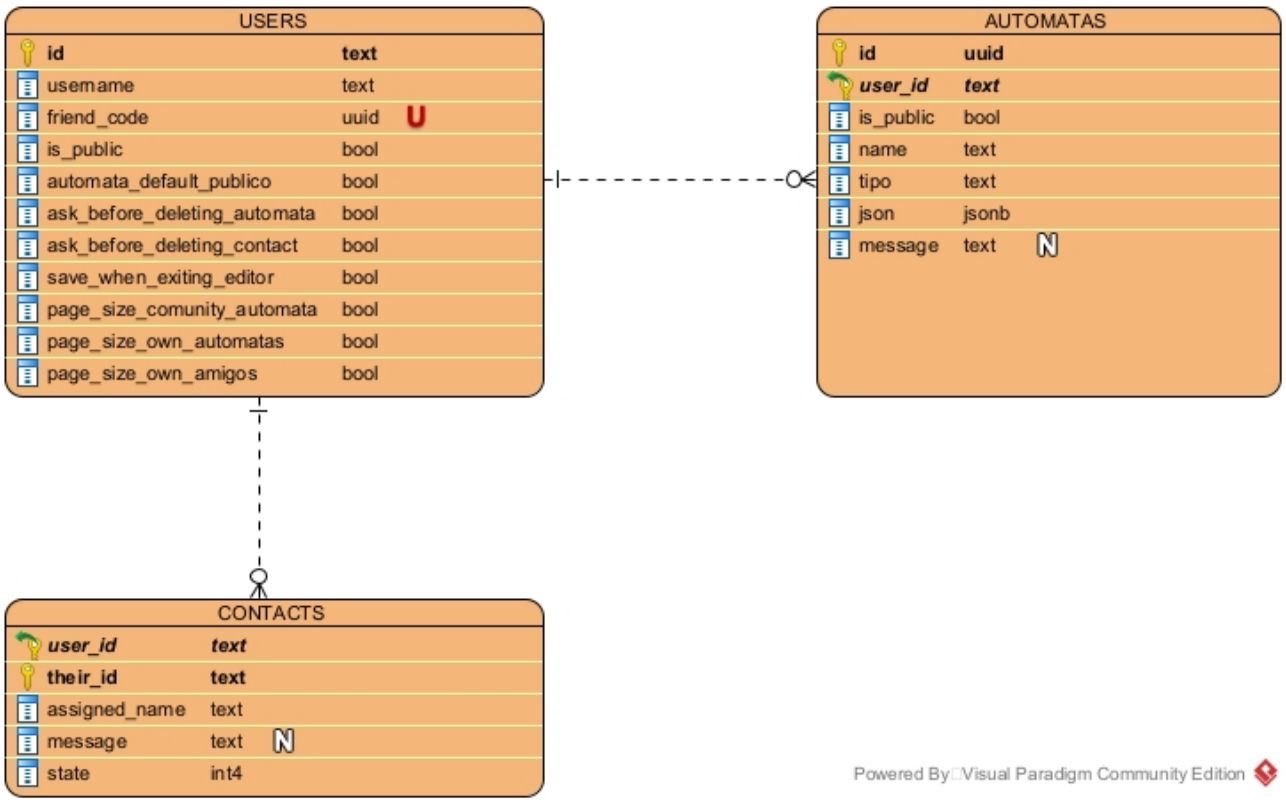

Figura 6.1: Diagrama del Modelo Entidad Relación de la Base de Datos

6.4. Diseño de la Interfaz de Usuario

Lo primero que se ha hecho es realizar unos bocetos de posibles interfaces de usuario que pueda satisfacer los requisitos funcionales planeados.

Para esto se ha determinado que una interfaz con tres vistas sería ideal.

La primera actuaría como hub, un sitio donde poder acceder al perfil de usuario, la lista de autómatas, amigos, etc. La segunda como lugar donde editar autómatas (editor de autómatas), editor de autómatas y la tercera un lugar donde simularlos (simulador de autómatas)

En base a estas tres vistas se ha realizado un boceto de la interfaz de usuario lo que ha dado lugar al primer prototipo de la aplicación. Este permite realizar si no todos, la mayoría de los casos de uso.

Tras desarrollar la mayoría de casos de uso, se ha dado a probar dicho prototipo a usuarios potenciales de la aplicación, pidiéndoles retroalimentación.

En base a la retroalimentación obtenida y a observaciones propias del uso de la aplicación se ha rediseñado parte de la interfaz, en concreto la que en el Primer Boceto es denominada Vista1, dando

lugar al Boceto 2.

Finalmente una vez toda la aplicación está terminada se ha vuelto a dar a probar y se ha vuelto a modificar la aplicación en base al retroalimentación, si bien solo se han aceptado cambios pequeños para esta.

Las siguientes secciones recogen los tres bocetos que se han realizado de la interfaz de usuario. Estos han sido realizando utilizando Excalidraw [22] y Paint [23] (para editar algunos detalles). Además los bocetos se encuentran brevemente comentados sobre la funcionalidad que implantarían y otros detalles.

6.4.1. Primer Boceto

Figura 6.2: Boceto 1: Vista1a

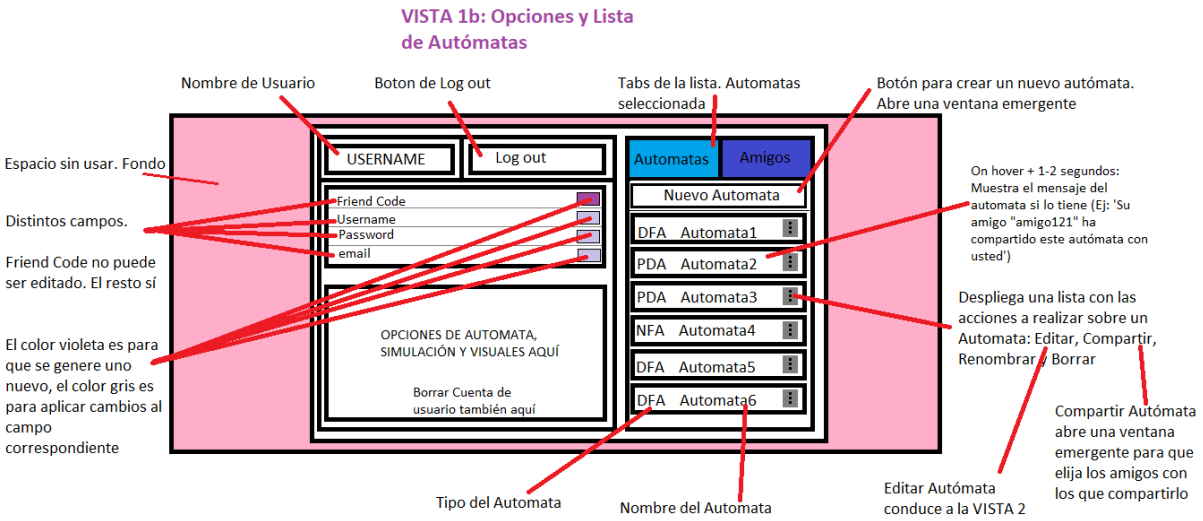

Figura 6.3: Boceto 1: Vista1b

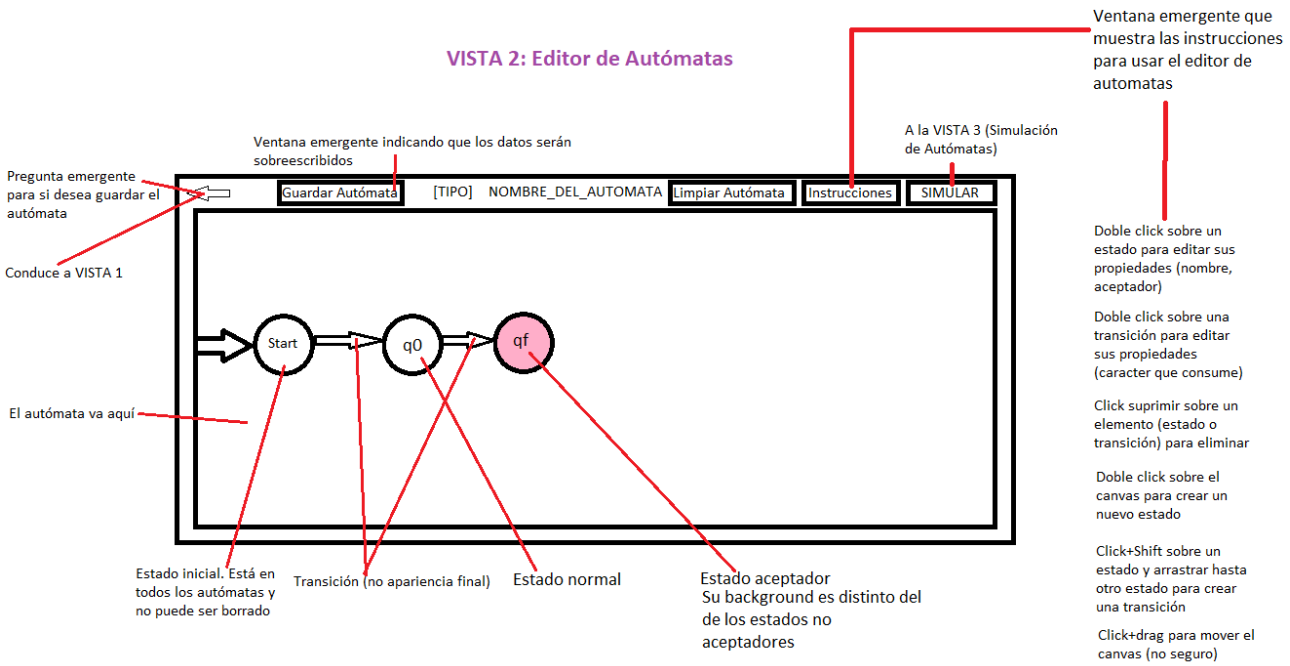

Figura 6.4: Boceto 1: Vista2

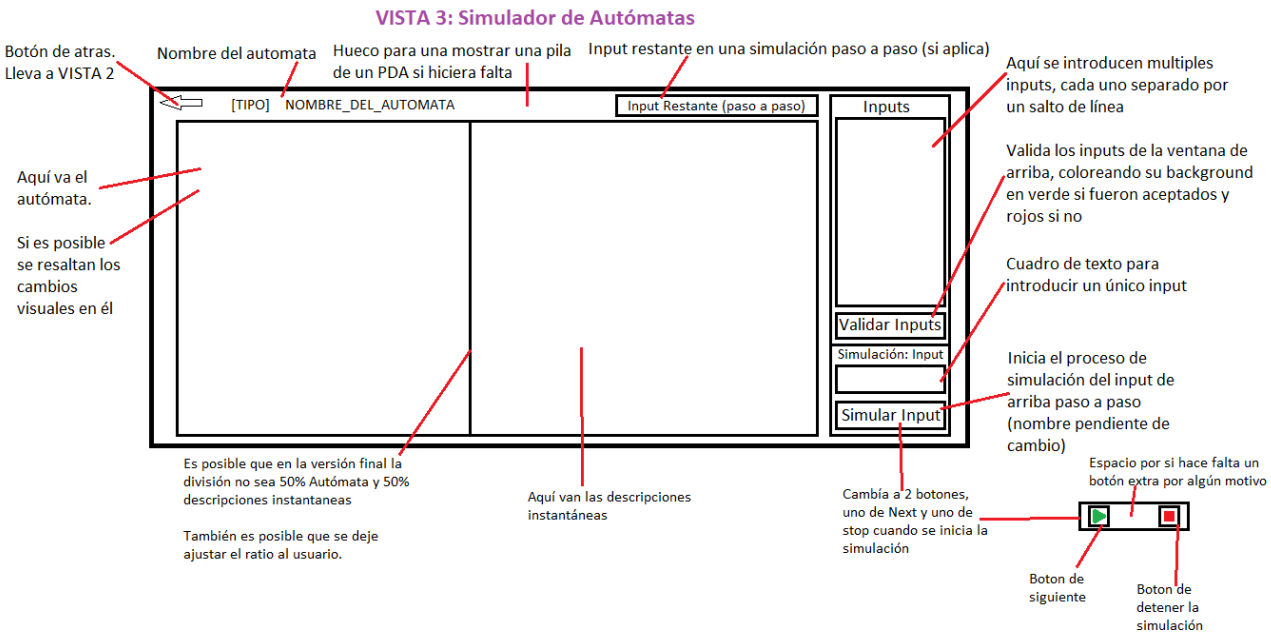

Figura 6.5: Boceto 1: Vista3

6.4.2. Segundo Boceto

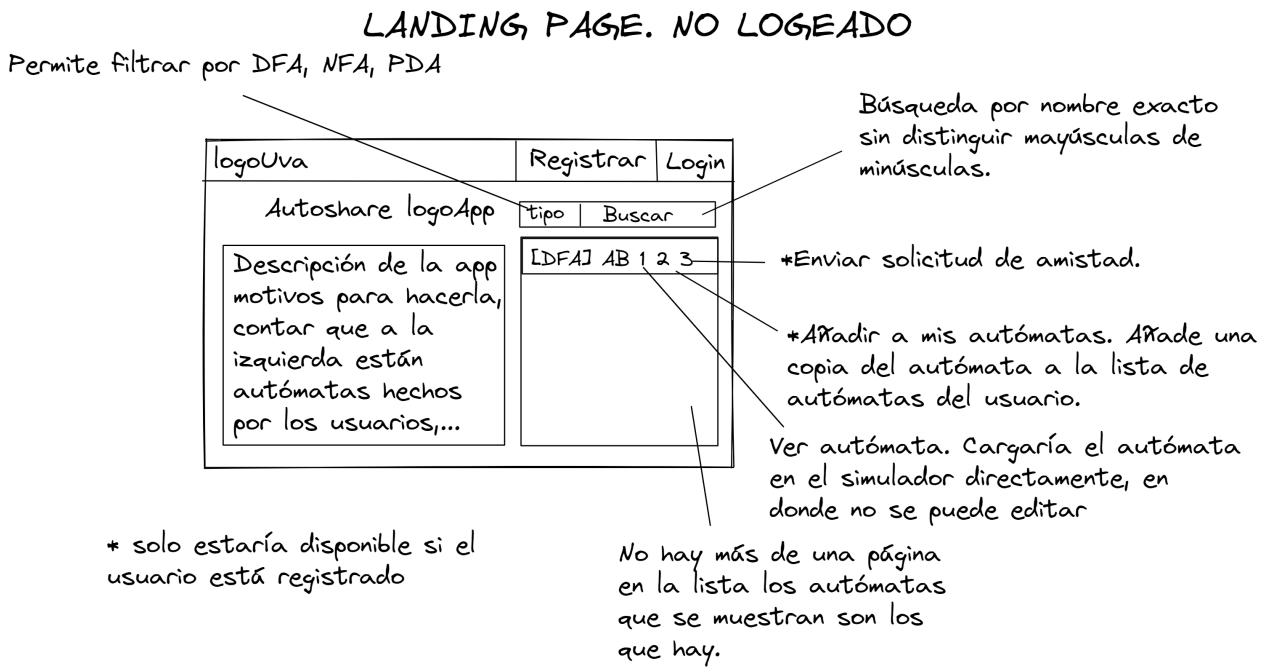

Figura 6.6: Boceto 2: Página Principal (Comunidad (Sesión no Iniciada))

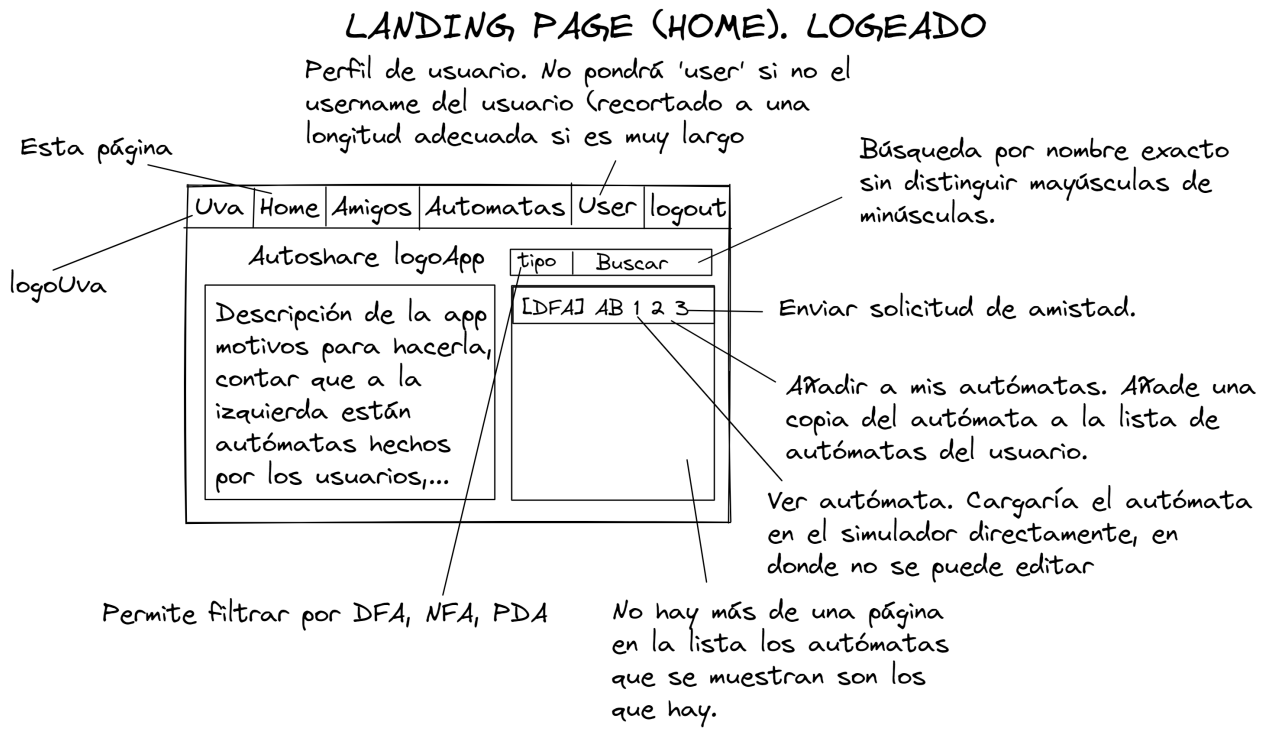

Figura 6.7: Boceto 2: Página Principal (Comunidad (Sesión Iniciada))

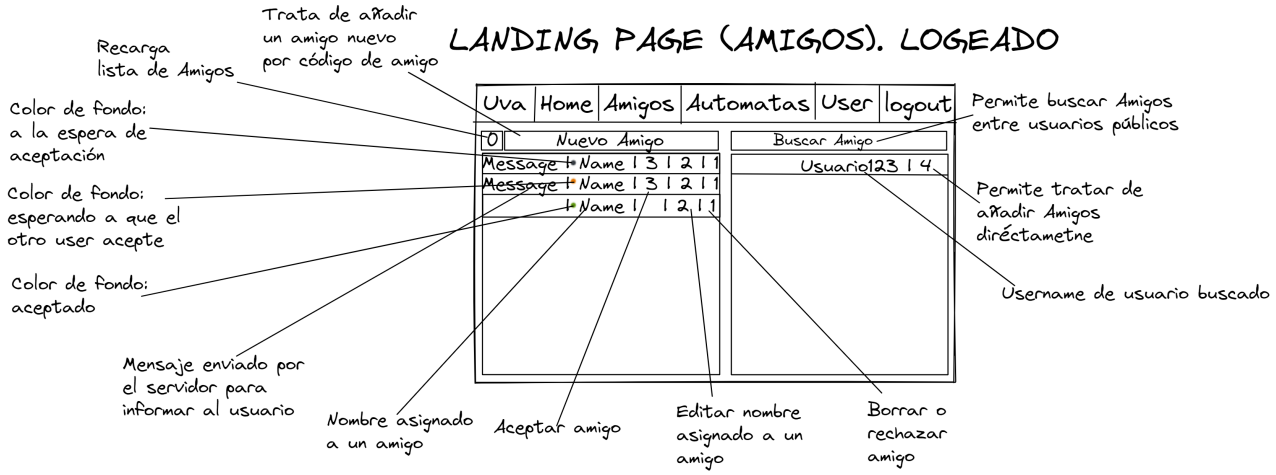

Figura 6.8: Boceto 2: Página Principal (Amigos)

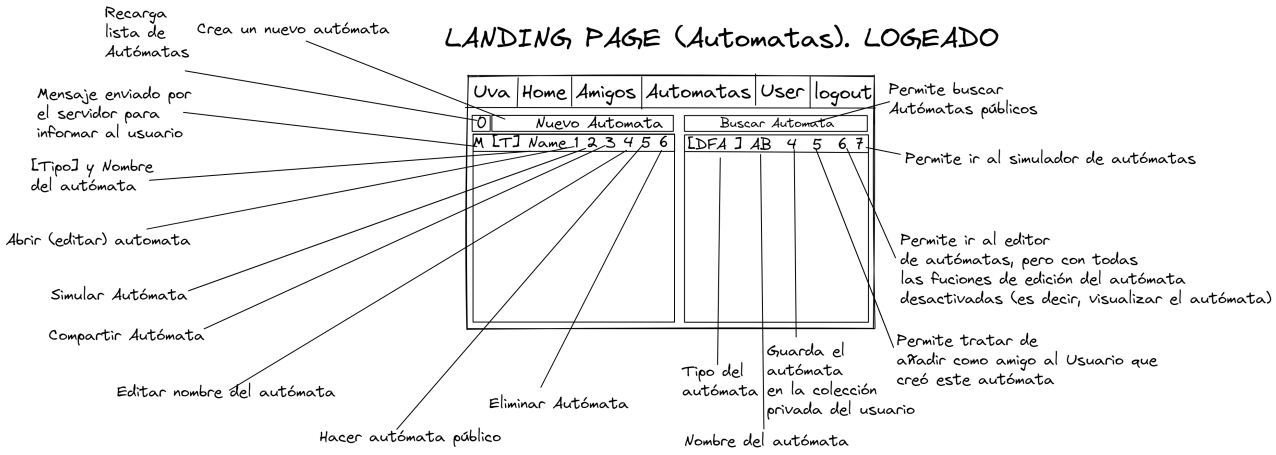

Figura 6.9: Boceto 2: Página Principal (Autómatas)

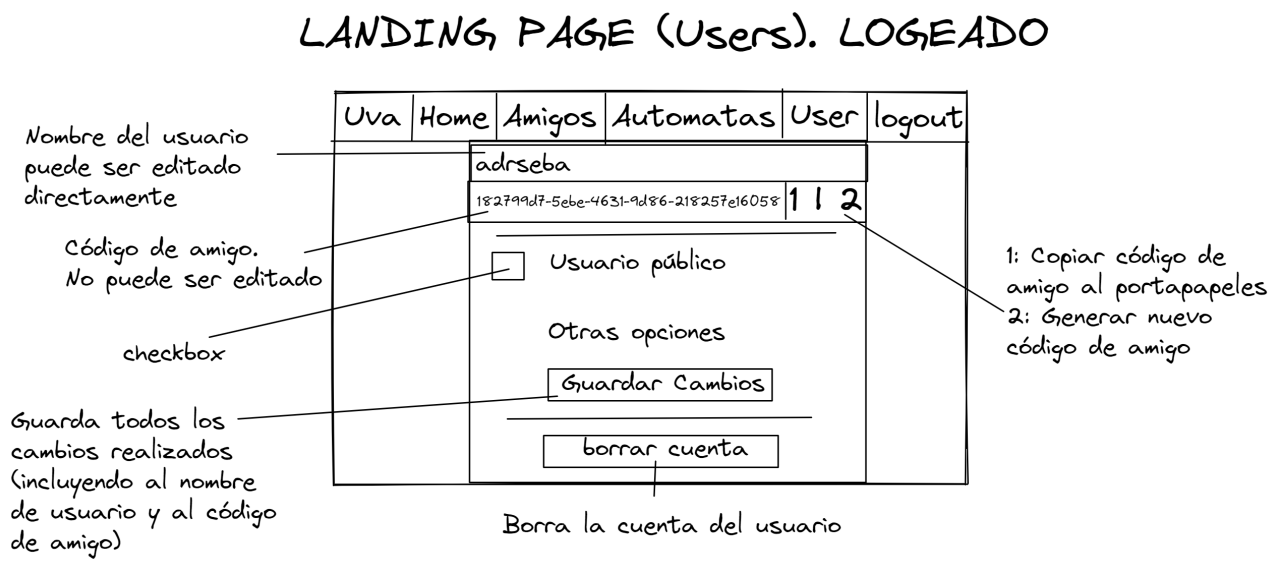

Figura 6.10: Boceto 2: Página Principal (Usuarios)

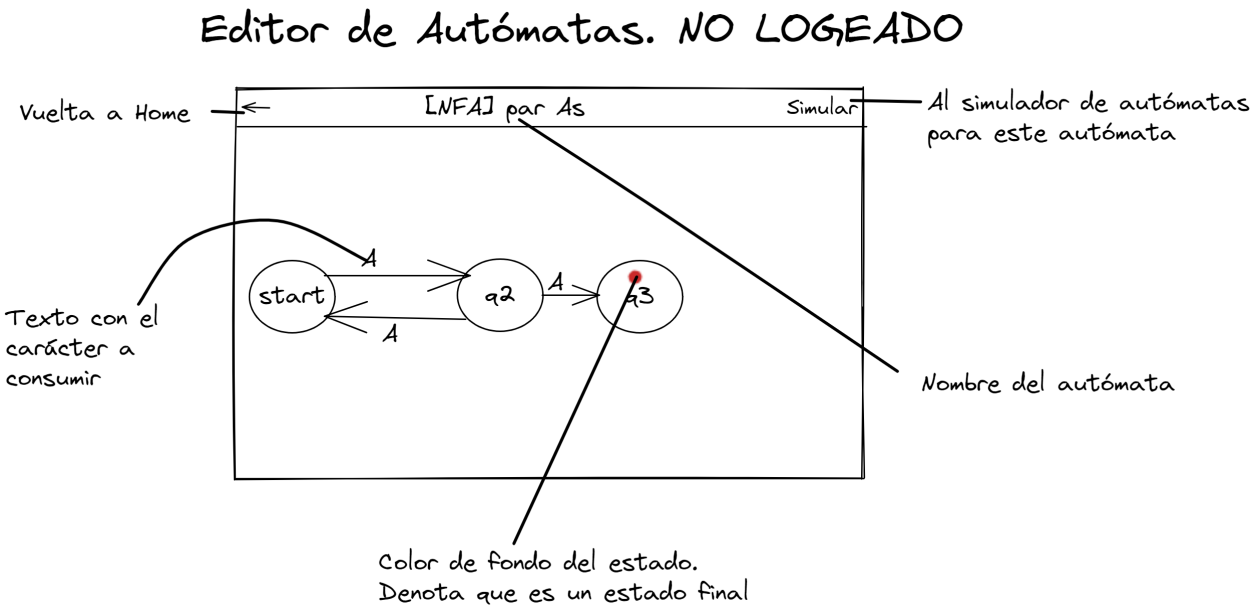

Figura 6.11: Boceto 2: Editor De Autómatas (Autómata No Editable)

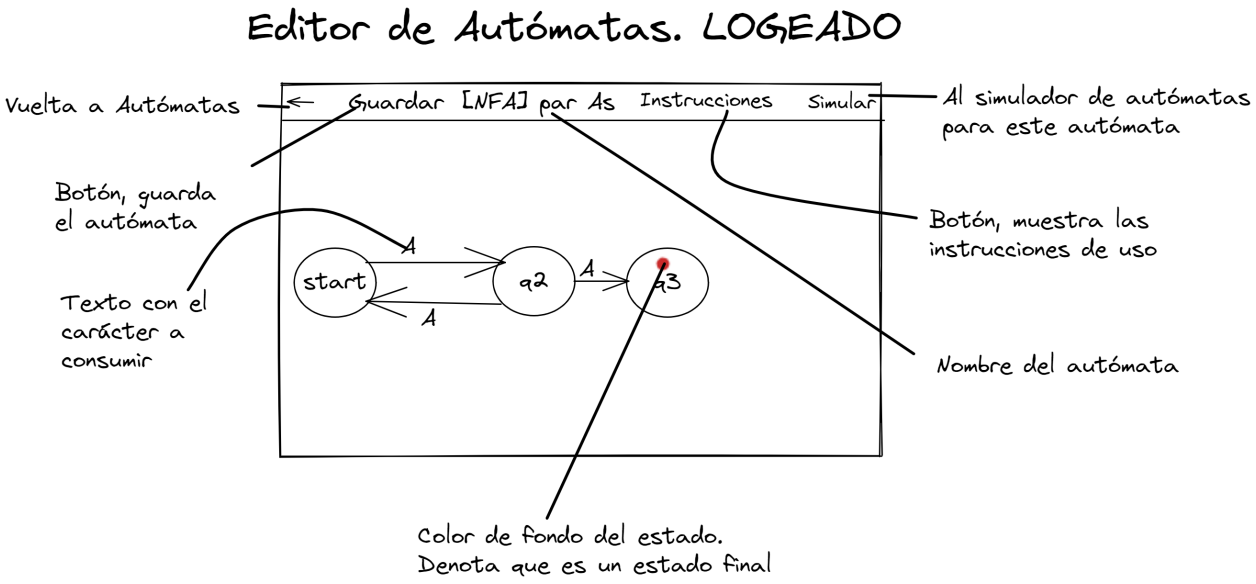

Figura 6.12: Boceto 2: Editor De Autómatas (Autómata Editable)

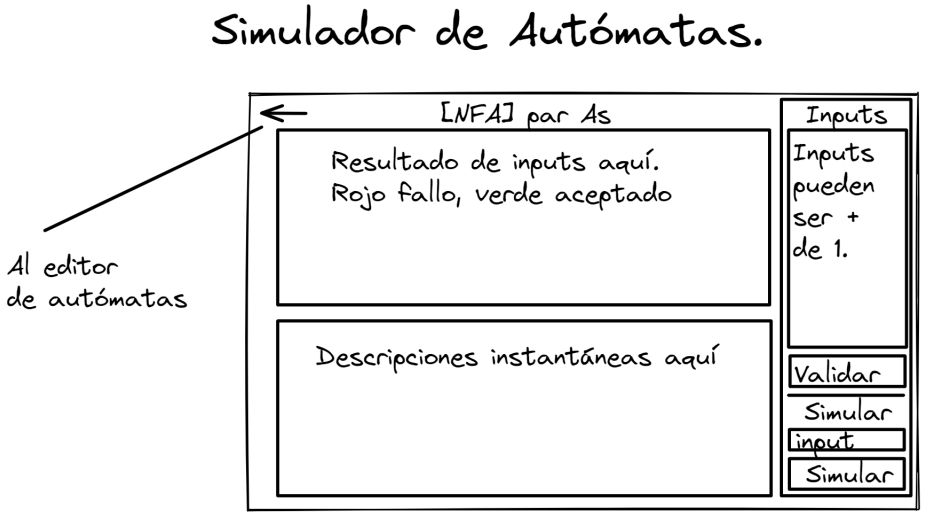

Figura 6.13: Boceto 2: Simulador De Autómatas

6.4.3. Tercer Boceto

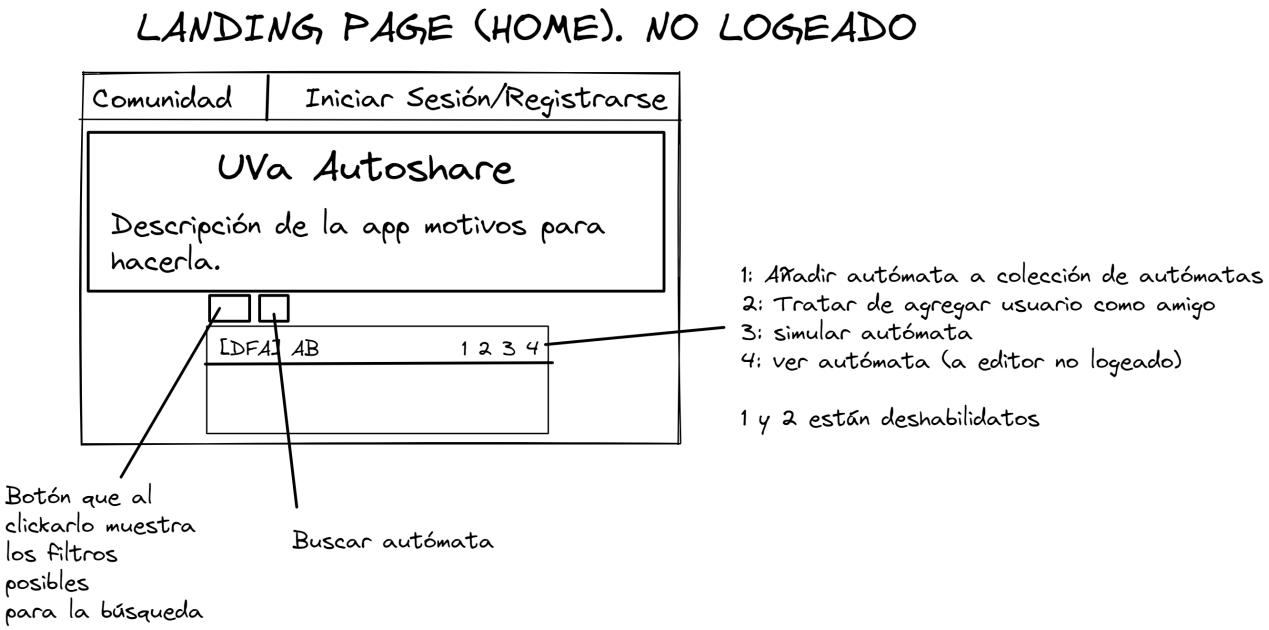

Figura 6.14: Boceto 3: Página Principal (Comunidad (Sesión no Iniciada))

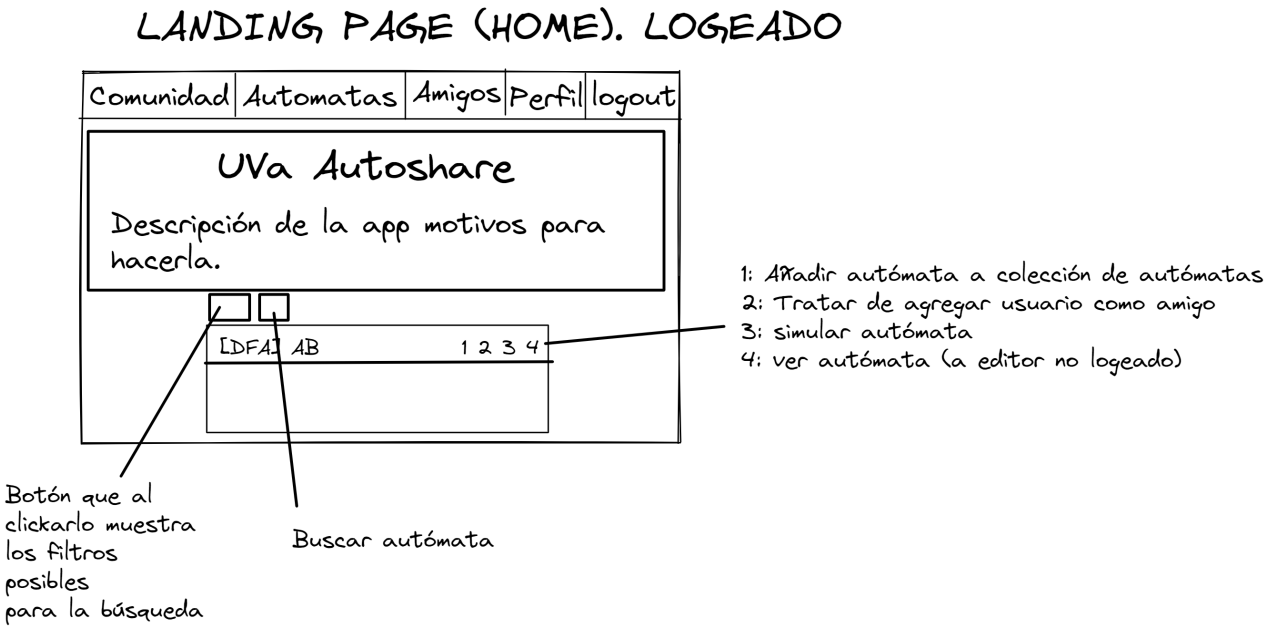

Figura 6.15: Boceto 3: Página Principal (Comunidad (Sesión Iniciada))

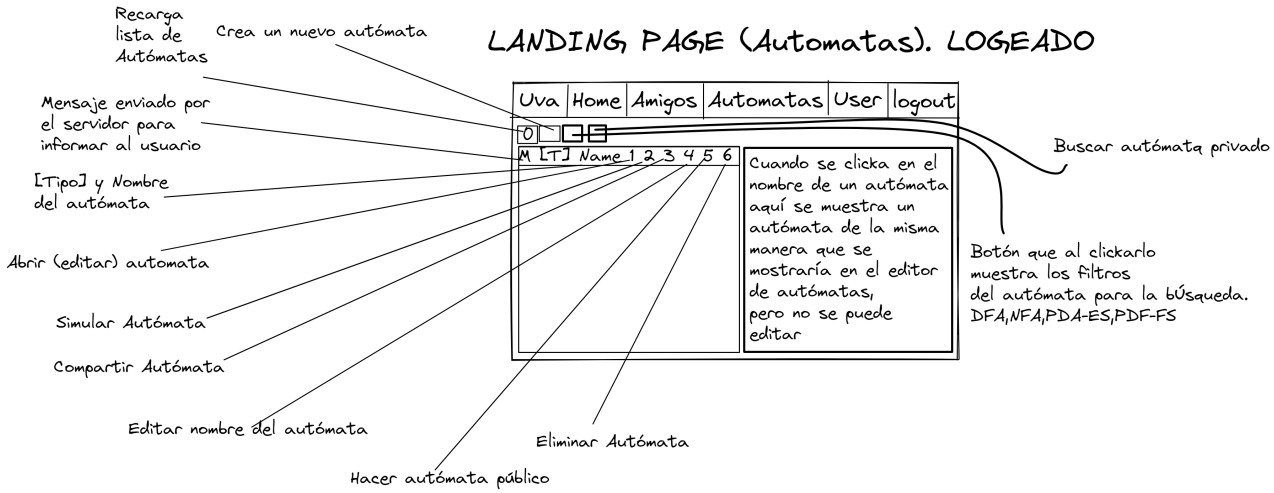

Figura 6.16: Boceto 3: Página Principal (Autómatas)

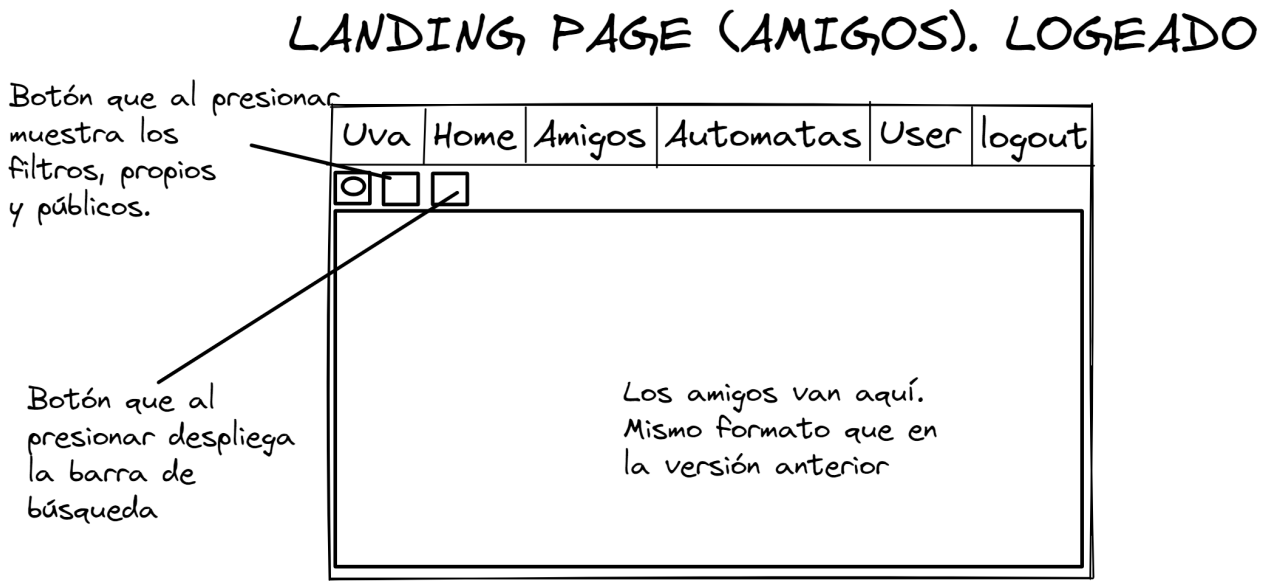

Figura 6.17: Boceto 3: Página Principal (Amigos)

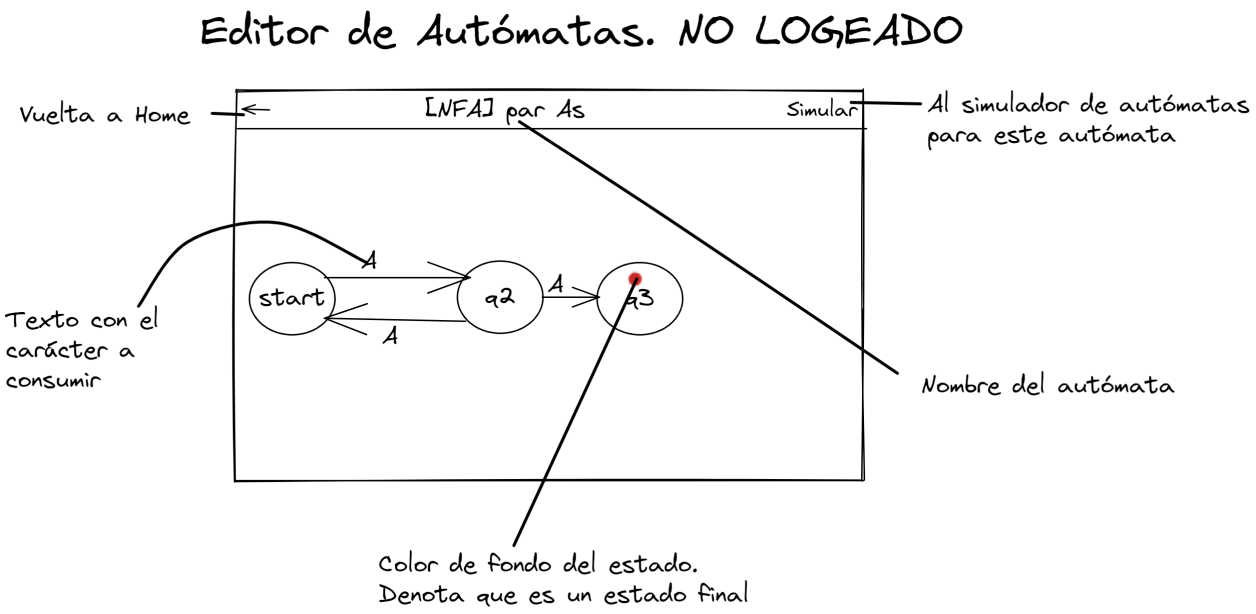

Figura 6.18: Boceto 3: Editor De Autómatas (Autómata no Editable)

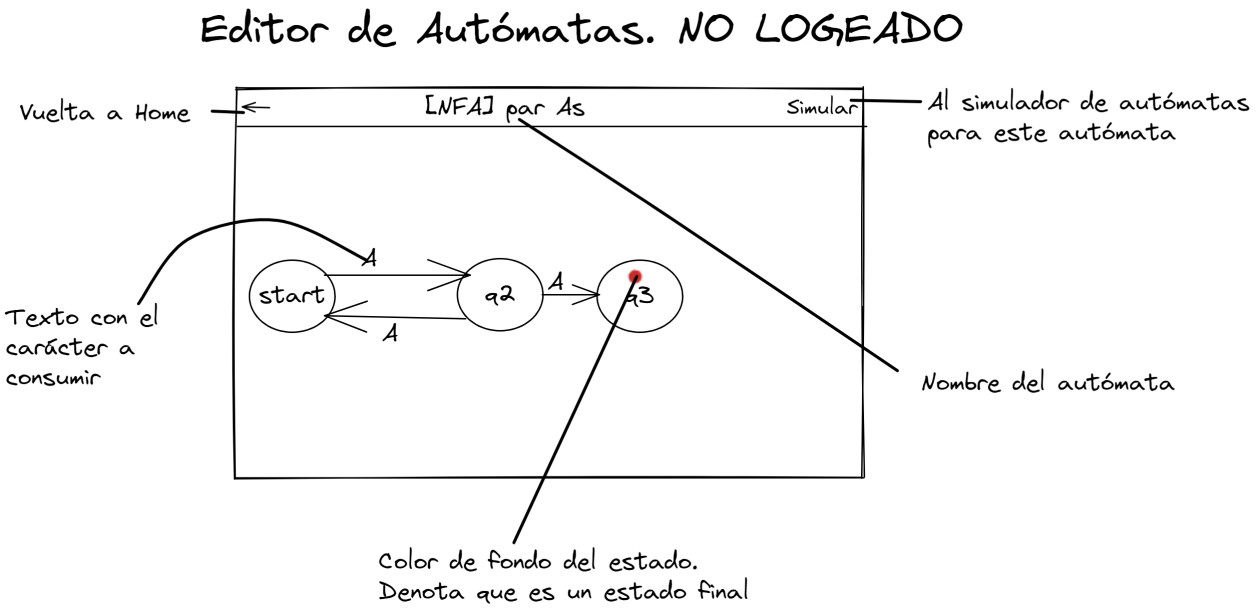

Figura 6.19: Boceto 3: Editor De Autómatas (Autómata Editable)

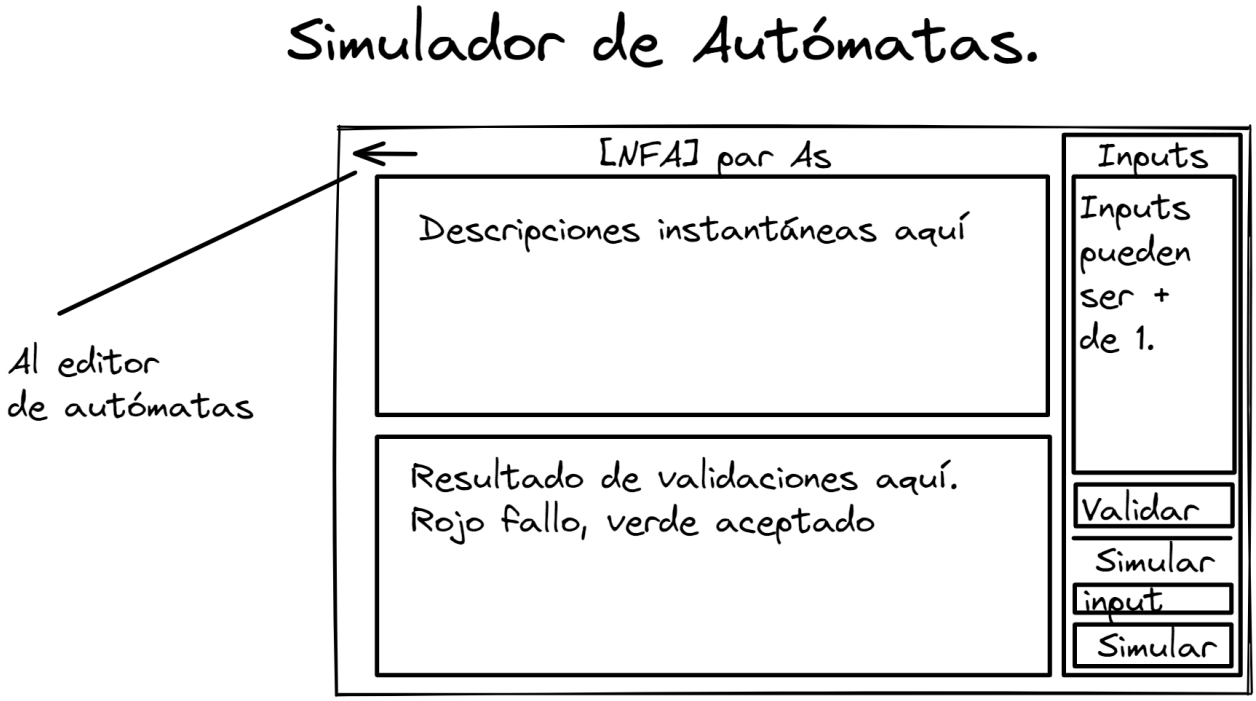

Figura 6.20: Boceto 3: Simulador De Autómatas

6.5. Diagramas de Secuencia

En esta sección se incluyen dos diagramas de secuencia para ilustrar el flujo de las operaciones.

El primer diagrama de secuencia es sobre el caso de uso RF3.10: Validar Cadenas de Entrada y sirve para ilustrar el flujo normal de la aplicación cuando no hay efectos secundarios (es decir, cunado no hay que hacer un request al servidor)

El segundo diagrama trata el caso de uso RF3.4: Buscar Autómatas y busca ilustrar el flujo de la aplicación cuando hay efectos secundarios, en este caso que es necesario consultar y obtener los autómatas pertinentes de la base datos.

Un detalle sobe los diagramas es no hay realmente objetos *reducer*, handler, request o watcher-saga si no que todos ellos son funciones. Se encuentran puestos así por claridad.

Los *reducers* están porque react-redux, el cual crea un estado global (store), se actualiza llamando a una función *reducer* que se encarga de actualizar una cierta parte de dicho estado global.

Por otro lado los handler son funciones en donde se han escrito los side-effects mientras que los request son las funciones que realizan requests a un servidor. Se han denominado y separado de esta manera para tener el código organizado y poder ser reutilizable en otras funciones.

Finalmente watchersaga es una función que intercepta todas las acciones y las redirige a los handlers en caso de que tengan side-effects, si no no hace nada.

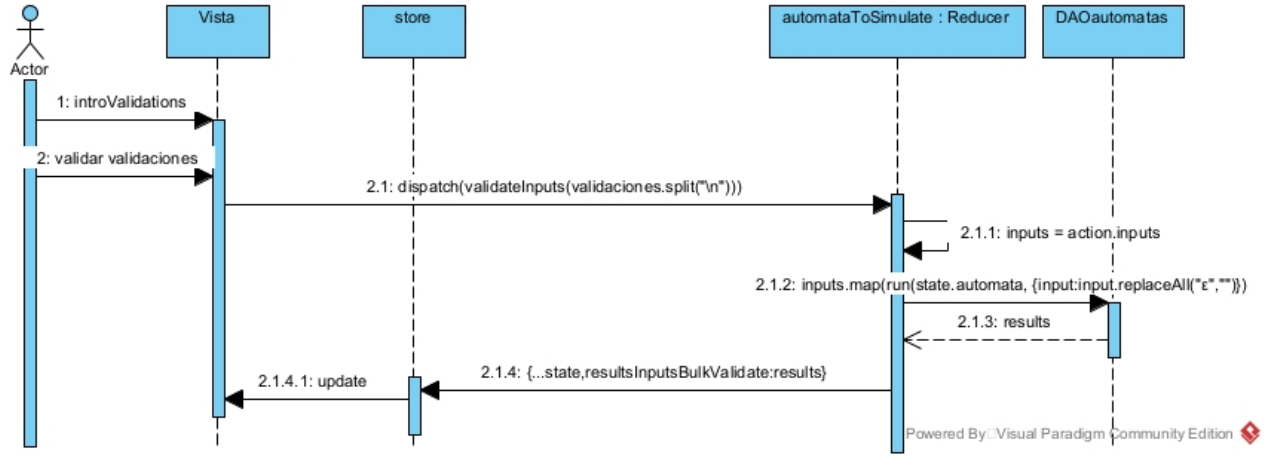

Figura 6.21: RF3.10: Validar Cadenas de Entrada

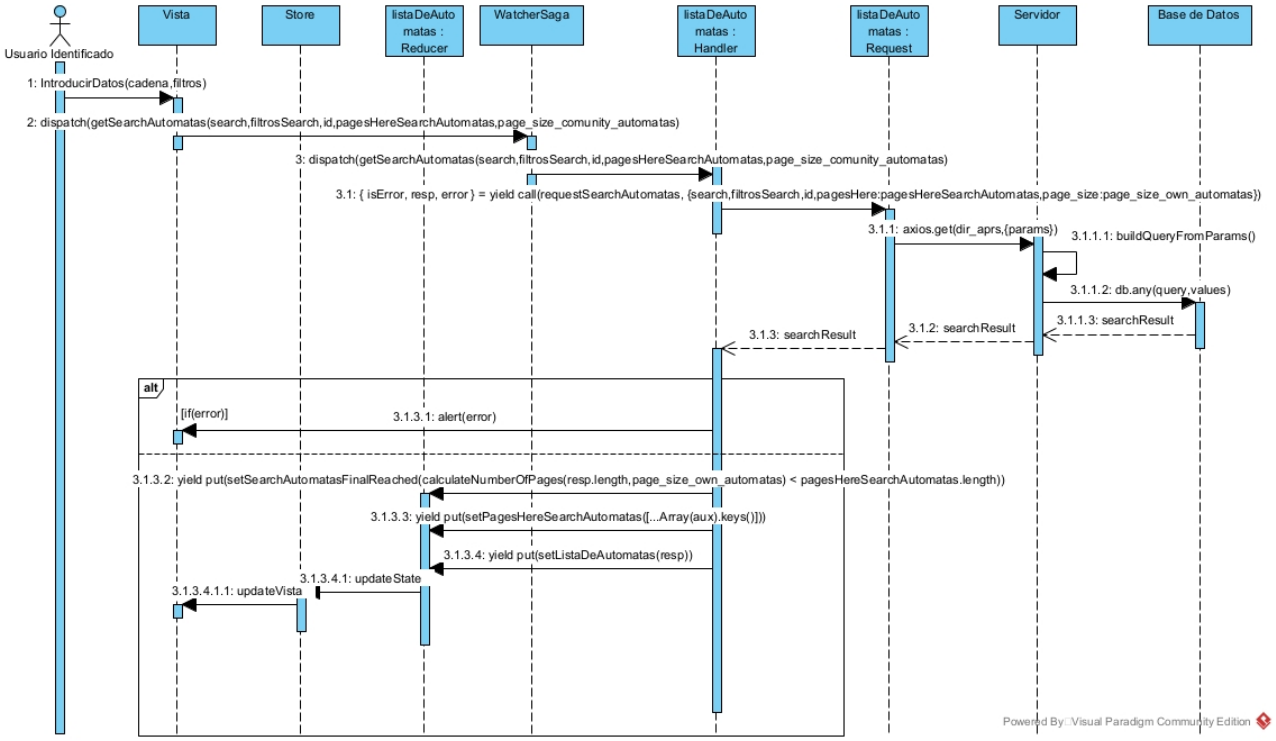

Figura 6.22: RF3.4: Buscar Autómatas

Capítulo 7

Implementación

7.1. Herramientas de Desarrollo

Existen un gran número de tecnologías y herramientas relacionadas que me permitirían conseguir los objetivos planteados para la aplicación.

Para la parte de Frontend han considerado en un principio Jquery [24], React [5], Angular[25] y Vue [26].

Jquery es una librería de Javascript que facilita enormemente la manipulación y el recorrido del DOM (Document Object Model). También simplifica llamadas Ajax, manipulación de CSS, métodos de eventos en HTML, efectos y animaciones y otras múltiples utilidades [24].

React es una librería de código abierto para frontend en Javascript desarrollada y mantenida por Facebook (aunque otros desarrolladores independientes también pueden colaborar) que sirve para construir interfaces de usuario y componentes [5].

Angular es un framework de código abierto para aplicaciones web basado en Type-script y desarrollada y mantenida por Google (aunque igual que en React desarrolladores individuales y/o corporaciones pueden colaborar) [25]. Angular se basa en el patrón MVC (Model-View-Controller) y en el enlace de datos bidireccional (two-way data binding), esto último significa que en la vista es posible definir el modelo y en el modelo modificar la vista [27].

Vue es un framework de código abierto para frontend en Javascript para construir aplicaciones de una sola página e interfaces de usuario desarrollado por un ex-trabajador de Google y actualmente mantenido por él y el resto de miembros de su equipo. [1]

Debido a que las 4 opciones se ajustaban a lo que necesitaba la manera de seleccionar entre ellas fue programar algunas aplicaciones básicas con ellas. Terminé eligiendo React debido a que fue con la que más fácil me resulto programarlas y con la que me más cómodo sentía. Instalé además las siguientes librerías para React: React-Redux, Redux-Saga y Axios.

[1] `https://en.wikipedia.org/wiki/Vue.js`

React es una librería de código abierto para frontend en Javascript desarrollada y mantenida por Facebook (aunque otros desarrolladores independientes también pueden colaborar).

React se utiliza para construir interfaces de usuario, permitiendo componer interfaces complejas a partir de componentes más simples.

Estos Componentes básicamente dicen a React lo que queremos que se muestre en pantalla y cuando sus datos cambian son re-renderizados por React. Cada componente tiene como parámetros unas propiedades (llamadas props).

Es posible interpretar estas propiedades como el estado del componente, por ejemplo: un botón con un prop llamado 'value' que se muestra en su etiqueta y que cada vez que se hace click en él incremente 'this.prop.value' en 1, actualizando el número que se muestra en su etiqueta.

Existe una limitación con esto sin embargo, y es que aunque pasar los props de un componente a otro es posible (bien sea hacia abajo en la jerarquía de componentes o hacía arriba), es muy lioso y confuso de escribir y entender. Para solucionar esta limitación he utilizado la librería de React-redux, una librería de middleware que enlaza React con Redux.

Redux permite almacenar un estado global en la aplicación que es simple de editar y fácilmente accesible por todos los componentes. Además de permitir separar la lógica que modifica el estado global de como se define el componente visualmente.

El estado global es almacenado en una única store, un objeto con forma de árbol, que solo puede ser modificado creando una action, un objeto que describe el cambio que le va a ocurrir al estado, y enviándolo (dispatch) a la store.

Para esto se usa una función *reducer*, la cuál recibe el estado actual y una action (función que devuelve un objeto json que tiene atributo 'type') y devuelve un nuevo estado. Sin embargo, a medida que una aplicación crece el tener un único *reducer* global puede ser problemático, así que conviene dividir el *reducer* en otros *reducers* más pequeños y específicos para cada componente.

De esta forma cada componente que pueda modificar el estado, y por tanto tenga actions, de alguna forma tiene que tener un *reducer* en donde se describe como sus actions modifican el estado de la aplicación.

Sin embargo, I igual que con React puro existe una limitación y esta es los side-effects. En ocasiones es necesario escribir datos en el servidor o traer datos de este y en base a eso modificar el estado de la aplicación. E igual que con React es posible sin usar librerías adicionales (excepto las necesarias para llamar al servidor), pero complica la legibilidad del código y no escala muy bien a aplicaciones grandes.

Para solucionar esto he utilizado la librería Redux-saga que se encarga de tratar con los side-effects de las actions.

Redux-saga es usada en mi aplicación para monitorizar todas las actions que son enviadas (dispatch-ed), esto lo hace usando un Objeto watcher-saga.

Watcher-saga es un objeto en el que es posible indicar tipos de acción y funciones para que se encarguen de ellas. Si una de las acciones que monitoriza (ya que es posible hacer que solo

monitorice parte de las acciones y no todas) coincide con alguna de las que tiene indicadas, tras que esta vaya al *reducer* (es posible asignarle a la action que simplemente devuelva el estado actual sin hacer cambios, actuando como si nunca hubiese ido al *reducer*), lanza la función que dicha action tenga asignada en el watcher-saga, pasándole la action como parámetro.

Esta función, que debe ser asíncrona, se encarga de ejecutar los side-effects y de enviar (dispatch) las actions necesarias para modificar el estado si esto es necesario. Es común para estos casos el tener una actions que sean, por ejemplo, 'POST_USER', 'POST_USER_SUCCESS', 'POST_USER_FAILURE' o 'FETCH_USER' para disparar y luego tratar los resultados de los side-effects. Para incorporar los resultados de los side-effects al estado en general la misma acción luego dispara otra acción.

La llamada actual al servidor es realizada por axios, una librería que facilita las llamadas a REST API.

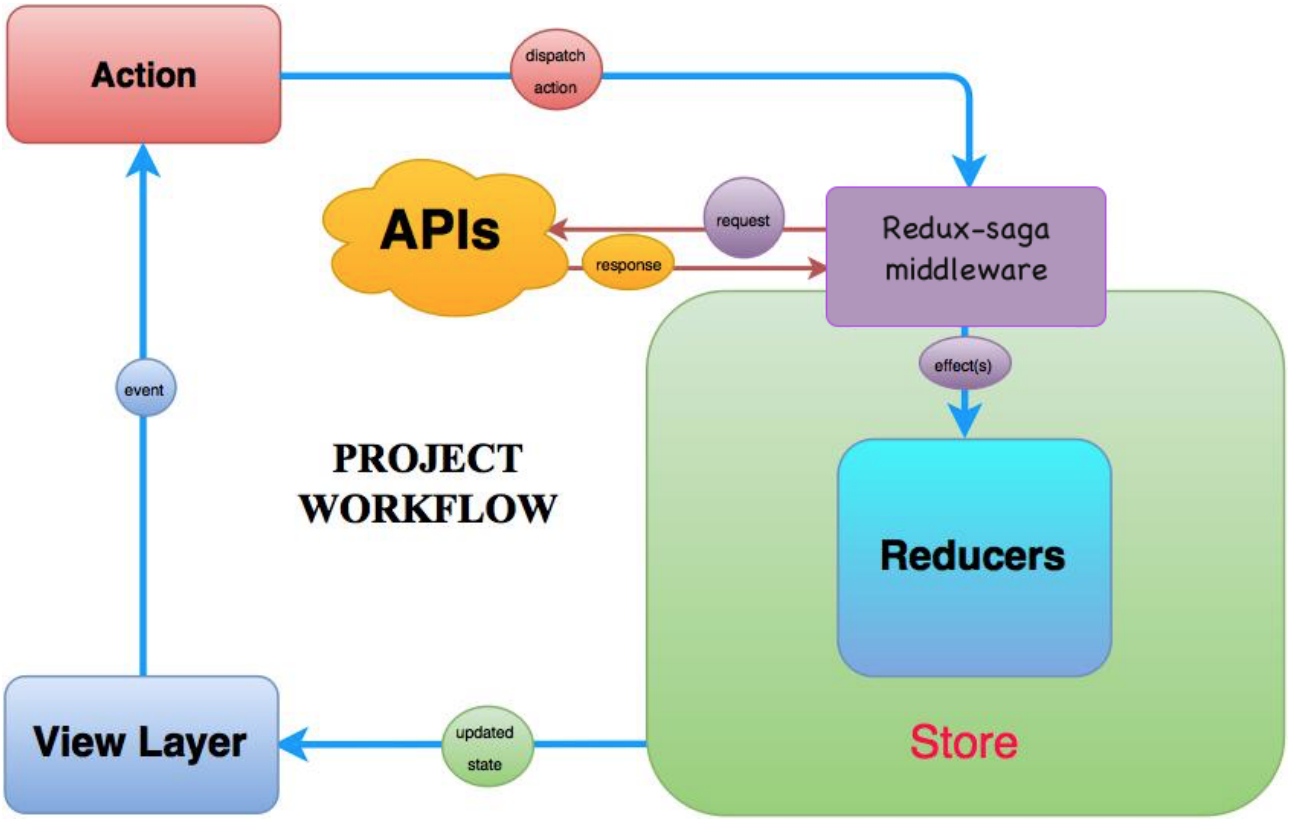

Figura 7.1: Flujo de la aplicación [28]

Finalmente además de las librerías mencionadas para el Frontend también he utilizado la librería material-ui [8].

De esta librería es de donde he sacado los componentes que luego he utilizado para componer las distintas vistas que forman la interfaz gráfica de usuario.

El Backend consiste de un servidor creado usando Node y, en concreto, Express así como una base de datos PostgreSQL. El motivo de escoger estás tecnologías con respecto a otras que ofrecen soluciones similares es que ya las había utilizado con anterioridad y estaba relativamente familiarizado con ellas.

Para comunicar el servidor node con la base de datos he utilizado la librería pg-promise.

Debido a que es necesario que los usuarios se identifiquen y registren para usar la aplicación, para que esto se efectué con seguridad he utilizado Okta [2] como proveedor de identidad para almacenar los email y password de los usuarios de manera segura.

Para el login he utilizado el okta Signin Widget (okta-signin-widget) [10] y para comunicar la aplicación con React para he utilizado las librerías okta-react, okta-auth-js. Las dos primeras para comunicar la aplicación con Okta, mientras que la última ha sido.

Otras librerías auxiliares que he utilizado son react-router-dom para redirigir a distintas direcciones dentro de la aplicación, jwt-decode para decodificar el token de acceso a okta y uuid para generar uuidv4.

Finalmente para la representación y edición de los autómatas de manera visual se ha utilizado la librería D3 [4] (versión v3 en concreto). Esta es una librería para Javascript, comúnmente usada para crear gráficos, permite enlazar datos al DOM y luego aplicar transformaciones sobre ellos.

7.2. Implementación

7.2.1. Gestión de Credenciales de Usuario

OKTA es utilizado para manejar el acceso a la aplicación a través de su sign-in widget. Gracias a OKTA el acceso a la aplicación se realiza de forma seguro, con los datos del usuarios almacenados procesados y almacenados de manera segura.

Para poder usar el widget se ha creado una cuenta de desarrollador en Okta de manera gratuita y se ha registrado ahí una aplicación.

En la aplicación creada en Okta se ha asignado como URI de redirección cunado se inicia sesión y cuando se cierra sesión la Página de Inicio. Esto es porque la Página de Inicio varía en función de si el usuario se encuentra con la sesión iniciada o no.

Además de esto se ha activado para la aplicación el auto registro ya que la configuración por defecto en las aplicaciones cuando se registra un usuario en Okta es necesario que alguien con los suficientes permisos, es decir yo en este caso, tendría que aprobar al usuario. Por razones de seguridad se pide que un usuario verifique el email cuando tiene que registrarse.

Finalmente también se han modificado los campos por defecto que se piden para registrase a nickname, email y contraseña (por defecto: primer nombre, apellido, email y contraseña).

Una vez un usuario inicia sesión en la aplicación a través de Okta, usando el middleware se obtiene su id de Okta que es usado como clave primaria (Okta garantiza su unicidad) para los usuarios en la base de datos.

7.2.2. Autómatas

Representación Visual

Un autómata, sea cual sea su tipo, es representado visualmente al usuario en forma de grafo. Para hacer esto se sigue la descripción de como representar un autómata en forma de grafo. Cada Estado del autómata es un nodo y cada transición del autómata es una arista. Los estados finales son marcados con otro color y el estado inicial es denotado mediante el nombre 'Start'.

Para evitar errores por parte del usuario, la aplicación no deja meter transiciones erróneas de forma que todos los autómatas generados siempre son correctos.

La implementación del grafo que se utiliza para representar al autómata está basado en la implementación de este grafo dirigido [2] interactivo. Aunque fue necesario ampliar su funcionalidad y modificarlo para que funcionase y se integrase correctamente en React.

La última parte fue complicada debido a que el estado global en React siempre debe ser actualizado de manera inmutable, esto es, reemplazando la referencia de los campos que se reemplazan, no solo cambiando su valor. Sin embargo, la implementación del Grafo desde la que se partió representa el grafo como un objeto mutable, y todas las actualizaciones sobre si mismo (como añadir nodo) mutan el grafo.

Para solucionarlo lo que se hace es no actualizar el estado del autómata a menos que se guarde o se vaya a salir de la vista del editor de autómatas de la aplicación. Permitiendo trabajar con el grafo de forma mutable y manteniendo la inmutabilidad del estado global.

Para integrar React y D3 se creo un Custom Hook partiendo de una respuesta en StackOverflow [3]. El objetivo del Hook es proveer una referencia inmutable al volver a renderizar del componente (es decir, que si se utiliza en un componente, y este por cualquier motivo se debe volver a renderizar esta referencia será igual a la que había antes), aunque sí lo hace si el componente se desmonta.

Simulación

Es posible tanto validar una cadena de entrada, como simularla paso a paso. Internamente ambas funciones se realizan de la misma manera, la única diferencia siendo la información que muestran al usuario. Validar la cadena solo devuelve al usuario el resultado final de la validación, si dicha cadena es aceptada o no. Simularla paso a paso va mostrando al usuario las descripciones instantáneas generadas y si estas son aceptadas o no, parando solo cuando no todas sean rechazadas o haya una aceptada.

Debido a que es posible que existan bucles infinitos en los PDA se ha elegido 10000 como número máximo de iteraciones para la validación directa. Esta restricción no existe para la simulación paso a paso.

[2] https://github.com/cjrd/directed-graph-creator

[3] https://www.pluralsight.com/guides/using-d3.js-inside-a-react-app

7.2.3. Reducers y Estado Global

La aplicación posee un Estado Global al que puede acceder cualquier componente. El Estado Global se encuentra dividido en múltiples partes, cada una con su propio *reducer*, que es la función que se encarga de actualizar el estado de dicha parte.

Estas partes son las siguientes:

- Autómatas en editor: autómata seleccionado por el usuario y abierto en el editor de autómatas.
- Autómatas a mostrar: autómata seleccionado por el usuario para visualizarlo, a diferencia de autómata en editor, aquí solo se contienen los parámetros necesarios para visualizar dicho autómata.
- Autómatas a simular: autómata seleccionado por el usuario para ser simulado. Solo se encuentran aquí los parámetros necesarios para simular el autómata así como para mostrar la información de la simulación de manera entendible al usuario (ejemplo: nombres que el usuario ha asignado a los nodos en lugar de su id interno)
- lista de autómatas: contiene la lista con todos los autómatas, tanto del usuario como los autómatas públicos que se buscan, así como atributos extra para permitir el correcto funcionamiento de las acciones relacionadas a estos.
- lista de amigos: contiene la lista con todos los contactos, tanto del usuario como los contactos públicos que se buscan, así como atributos extra para permitir el correcto funcionamiento de las acciones relacionadas a estos.
- dialogo de transición: contiene un único parámetro boolean que determina si el dialogo de transición, dialogo que se abre para dejar que el usuario introduzca una transición nueva, está abierto o no. Este estado fue necesario debido a que es necesario abrirlo desde un componente hermano.
- home: contiene un único parámetro que determina que pestaña en la página inicial está abierta
- perfil de usuario: contiene todos los parámetros relacionados directamente con el usuario usando la aplicación.

Existe información repetida en este estado, el motivo de esto es la legibilidad del código así como su facilidad de uso por parte de la aplicación.

De estos *reducers* solo lista de autómatas, lista de amigos y perfil de usuario contienen acciones con side-effects. Lista de autómatas contiene todos los side-effects relacionados con los autómatas, traerlos del servidor, modificarlos, eliminarlos, crearlos, compartirlos con otros usuarios y buscarlos. Lo mismo sucede para lista de amigos pero para los contactos del usuario en lugar de para los autómatas.

Perfil de usuario por otro lado solo contiene side-effects de traer los datos relacionados con el usuario (id, nombre de usuario, código de amigo y opciones de usuario (número de autómatas por página en lista de autómatas, etc.)) y modificarlos, pero no crearlos o borrarlos.

Por facilidad se ha dividido las funciones que se encargan de los side-effects en dos, handlers y requests. Requests realizan un HTTP request al servidor para que realice una acción mientras que los handlers se encargan de las consecuencias de dicha acción, incluido el lanzar otra acción que actualice el estado del autómata en función a los resultados del request.

7.2.4. Servidor y Base de Datos

El servidor se utiliza como se utilizaría el modelo en un patrón modelo-vista-controlador. Este recibe HTTP requests y, en base al request y datos que se les pasa, realiza una operaciones otra sobre la base de datos.

El servidor es también el responsable de realizar requests a la API de OKTA usando un token de autorización (en este caso el mío, ya que al ser el administrador tengo los permisos necesarios para eliminar usuario de mi registro de OKTA) para dar de baja cuentas de usuario.

Todas las operaciones que realiza el servidor quedan registrados en un log para poder encontrar y determinar los fallos en caso de que sucedan.

La base de datos es usada para almacenar datos que deben persistir durante múltiples sesiones de un usuario (como los autómatas por ejemplo).

Capítulo 8

Pruebas

Para comprobar la validez del código de la aplicación es necesario realizar pruebas o tests que permitan comprobar la integridad del código generado. Dependiendo de lo que dichas Pruebas comprueben estás se pueden clasificar de distintas maneras:

- Pruebas de Unicidad [29]: Comprueban la validez de una función o clase. En general esto se realiza introduciendo a la función a comprobar diferentes parámetros de entrada, y la salida esperada. Si el resultado de la función con el esperado coincide esa prueba da bien. Suelen realizarse con ayuda de un librería y lanzarse siempre antes de compilar la aplicación para comprobar que no hay errores en las funciones, lo que permite un desarrollo escalable y sostenible.
- Pruebas de Integración [30]: Contrario a las Pruebas de Unicidad que solo comprueban el compartimiento de una pieza del código aislado del resto las Pruebas de Integración comprueban el conjunto, es decir, múltiples funciones o clases, denominados módulos, trabajando en conjunto con un objetivo.
- Pruebas del Sistema [31]: Pruebas realizadas sobre la funcionalidad de todo el sistema en conjunto. Su objetivo es verificar que que la aplicación o sistema desarrollado cumpla todos los requisitos especificados que se esperan de ella.
- Pruebas de Aceptación [32]: Estas pruebas se realizan con usuarios, tanto usuarios normales como con expertos, de la aplicación y tienen como objetivo el verificar la aplicabilidad de la aplicación. Es común recibir retroalimentación de dichos usuarios que realizan las pruebas y alterar la aplicación en consecuencia.

Aquí las Pruebas de Unicidad han sido encapsuladas en las Pruebas de Integración debido a que se han realizado numerosas pruebas de esta, quedando ya cubiertas.

Los resultados de las pruebas pueden ser dos, OK o KO. OK indica que la prueba ha sucedido como se esperaba, no ha habido ningún error inesperado y el resultado esperado de la prueba o sus objetivos se han cumplidos. KO por otro lado indica lo contrario, es decir, que la prueba ha fallado por algún motivo.

Una herramienta utilizada para realizar las pruebas ha sido emailfake [33]. Emailfake es un generador de dummy mails, es decir, emails falsos que serán eliminados al cabo de cierto tiempo. Esto

permite registrar nuevas cuentas en la aplicación fácilmente sin tener que estar creando múltiples correos reales, lo cual es ideal para probar casos que requiere que haya múltiples usuarios en la aplicación o simplemente para probar el registro de usuarios múltiples veces.

8.1. Pruebas de Integración

Para estas pruebas se han considerado los siguientes módulos, la suma de los cuales compone la aplicación.

- Lista de Amigos: Lista donde se muestra al usuario sus amigos. También se puede usar para buscar usuarios públicos.
- Lista de Autómatas: Lista donde se muestran al usuario sus autómatas. Desde aquí se realizan la mayoría de acciones relacionadas con un autómata tratándolo como entidad.
- Lista de Perfil de Usuario: Aquí se muestran las opciones de configuración actuales del usuario, su nombre de usuario y su código de amigo.
- Editor de Autómatas: Lugar donde se editan y guardan autómatas.
- Simulador de Autómatas; Lugar donde se provee al autómata cadenas de entrada para ver sus resultados.
- Servidor: Servidor de la aplicación, actúa como controlador y comunica la página web con la base de datos.
- Base de Datos: Lugar de almacenamiento de las opciones de configuración, autómatas y amigos de los usuarios.
- Okta Sign-in Widget: Widget facilitado por OKTA para que sea fácil registrarse e iniciar sesión. Es la forma de hacer esto en la aplicación
- OKTA: Verificador de identidad. Registra y almacena los emails y contraseñas de los usuarios.

A continuación se muestran tablas donde se resume el contenido y resultado de las pruebas realizadas, indicando que módulos son afectados, el tipo de prueba realizada, los objetivos a conseguir efectuando dicha prueba y el resultado de la misma (OK/KO).

Módulos	OKTA Sign-in Widget OKTA Servidor Base de Datos
Prueba	1. Se crea una cuenta de usuario y se verifica en OKTA que la cuenta esta creada 2. Se inicia y se cierra la sesión 3. Se trata de recuperar la contraseña de dicha cuenta y se inicia sesión de nuevo 4. Se borra la cuenta y se verifica en OKTA que la cuenta este borrada.
Objetivos	1. Se registra a un usuario en OKTA correctamente 2. Se elimina a un usuario elimina de OKTA correctamente 3. Se puede recuperar la contraseña de un usuario
Resultados	OK

Tabla 8.2: Prueba de Integración PI-1

Módulos	OKTA Sign-in Widget OKTA Servidor Base de Datos
Prueba	1. Se crea una cuenta de usuario 2. Se inicia sesión por primera vez con dicha cuenta 3. Se mira en la base de datos que id tiene el usuario. 4. Se crean entradas falsas para autómatas y contactos (usando los datos de usuario falsos) para la cuenta de usuario creada (usando su user_id). 5. Se cierra y se vuelve a iniciar sesión. 6. Se borrar la cuenta de usuario.
Objetivos	1. Al iniciar sesión por primera vez se crea una entrada para el usuario en la base de datos con 'user_id' el id del usuario en OKTA. 2. Al iniciar sesión se traen todos los datos del usuario del Servidor a la página web. 3. Al borrar una cuenta se eliminan los datos relacionados (amigos y autómatas) de dicha cuenta de la base de datos.
Resultados	OK

Tabla 8.4: Prueba de Integración PI-2

Módulos	Lista de Amigos Servidor Base de Datos
Prueba	1. Se crean dos cuentas de usuario (*userA* y *userB*) y se inicia sesión con ellas 2. Se inicia sesión en ambas, pero desde diferentes dispositivos, y se hace pública la cuenta de *userA*. 3. *userB* trata de buscar a *userA*. 4. *userA* busca a *userB* y le trata de agregar como amigo. 5. *userB* acepta a *userA*. 6. *userA* recarga la lista de amigos. 7. *userA* elimina a *userB* como amigo. 8. *userB* recarga la lista de amigos. 9. Se eliminan las cuentas de *userA* y *userB*.
Objetivos	1. Se verifica que solo se pueden buscar usuarios públicos 2. Se verifica que se reciben o eliminan amigos de la lista de amigos correctamente cuando se recarga la lista de amigos de forma que esta refleje el estado en la base de datos. 3. Se verifica que al eliminar un amigo la otra entrada en la base de datos es eliminada también para el otro usuario. 4. Se verifica que al agregar un amigo es creada otra entrada en la base de datos para el otro usuario 5. Se verifica que al actualizar el state de un amigo este cambio es reflejado en la base de datos correctamente.
Resultados	OK

Tabla 8.6: Prueba de Integración PI-3

Módulos	Lista de Amigos Perfil de Usuario Servidor Base de Datos
Prueba	1. Se crean una cuenta de usuario 2. Se puebla la base de datos con cientos de usuarios, todos ellos públicos. 3. Se inicia sesión en la cuenta y se buscan usuarios públicos introduciendo ” como termino de búsqueda 3. Se comprueba que solo se ha recibido cierta parte de los usuarios y no la lista entera de la base de datos. 4. Se accede al perfil de usuario y se cambia el número de amigos por página. 5. Se vuelve a la lista de amigos cargando de nuevo la lista de amigos. 6. Se eliminan la cuenta del usuario.
Objetivos	1. Se verifica que solo se recibe cierta parte de los usuarios de una vez y no todas las entradas que hay en la base de datos 2. Se verifica que al cambiar el número de amigos por página este cambia en consecuencia
Resultados	OK

Tabla 8.8: Prueba de Integración PI-4

Módulos	Lista de Amigos Perfil de Usuario Servidor Base de Datos
Prueba	1. Se crean dos cuentas de usuario, *userA* y *userB* 2. Se inicia sesión con ambas cuentas en distintos dispositivos, dejando *userB* como usuario privado 3. Se anota el código de amigo de *userB*. 3. Usando dicho código *userA* trata de agregar como amigo a *userB* 4. Se deja en blanco el alias del amigo 5. *userB* recarga su lista de amigos y elimina a *userA* de ella, rechazando el ser su amigo 6. *userB* obtiene un nuevo código de amigo 7. *userA* trata de agregar a *userB* de nuevo usando su código de amigo antiguo 8. Se borran las cuentas de *userA* y *userB*
Objetivos	1. Se verifica que al cambiar un código de amigo la base de datos genera un código nuevo y el antiguo se vuelve inútil 2. Se verifica que se puede utilizar un código de amigo para agregar a otro usuario 3. Se verifica que es posible agregar a usuarios privados mediante código de amigo
Resultados	OK

Tabla 8.10: Prueba de Integración PI-5

Módulos	Lista de Autómatas Servidor Base de Datos
Prueba	1. Se crean una cuenta de usuario y se inicia sesión. 2. Se crea un autómata, se marca como público y se busca en la lista de autómatas públicas. 3. Se cambia el nombre del autómata y se vuelve a buscar. 4. Se borra el autómata y se vuelve a buscar 8. Se eliminan la cuenta del usuario.
Objetivos	1. Se verifica que cuando se crea un autómata este se almacene en la base de datos 2. Se verifica que solo autómatas públicos puedan buscarse 3. Se verifica que si cambia el nombre del autómata este se actualice en la base de datos, y también para buscarse 4. Se verifica que cuando se borra un autómata este sea eliminado de la base de datos
Resultados	OK

Tabla 8.12: Prueba de Integración PI-6

Módulos	Lista de Autómatas Servidor Base de Datos
Prueba	1. Se crean crean dos cuentas de usuario, *userA* y *userB* 2. Se puebla la base de datos con cientos de autómatas, uno perteneciendo a *userB* y el resto a *userA*. 3. Se inicia sesión en ambas cuentas en diferentes dispositivos y *userA* y *userB* se agregan como amigos. 4. *userB* comparte el autómata con *userA* 5. *userA* busca el autómata en su lista de autómatas 6. *userA* cambia el nombre al autómata 7. Se eliminan la cuenta del usuario.
Objetivos	1. Se verifica que un autómata puede ser compartido con otro usuario y que tiene una entrada distinta en la base de datos que el autómata compartido, de forma que si uno es modificado el otro no lo es. 2. Se verifica que se carga un número limitado de autómatas en la lista de autómatas en lugar de todos de una vez. 3. Se verifica que a pesar del punto anterior, se puede buscar un autómata que no estaba cargado originalmente
Resultados	OK

Tabla 8.14: Prueba de Integración PI-7

Módulos	Editor de Autómatas Simulador de Autómatas Servidor Base de Datos
Prueba	1. Se crean una cuenta de usuario y se inicia sesión 2. Se crea un autómata 3. Partiendo de dicho autómata inicial se edita hasta tener un autómata que venga analizado en Introducción a la teoría de autómatas, lenguajes y computación [12] y se guardan cambios 4. Se sale del editor de autómatas y se vuelve a entrar, 5. Se procede al Simulador de autómatas y se proceden a introducir las cadenas de entrada con las que se explica el autómata editado en Introducción a la teoría de autómatas, lenguajes y computación [12] 6. Se vuelve al Paso 3 tomando como autómata inicial el actual. Paso 6 se realiza para varios ejemplos del libro 7. Se vuelve al paso 2, pero se cambia el tipo de autómata. Esto se hace hasta que se hayan probado todos los tipos de autómatas posibles 8. Se eliminan la cuenta del usuario.
Objetivos	1. Se verifica cada vez que se guarda que se actualiza la entrada del autómata en la base de datos 2. Cada vez que se actualiza la entrada del autómata en la base de datos se comprueba que el autómata representado actualmente en el editor coincide exactamente con el que está almacenado 3. Se comprueba que el autómata no es solo actualizado en el editor de autómatas si no también en el resto de la aplicación, sin necesidad de hacer una llamada al servidor. 4. Se comprueba que cuando se edita (añadir/eliminar estados y añadir/eliminar transiciones, des/marcar estados como finales, etc.) el grafo en el editor de autómatas estos cambios se reflejan en el autómata como tal. 5. Se verifica que las descripciones instantáneas son generadas correctamente.
Resultados	OK

Tabla 8.16: Prueba de Integración PI-8

8.2. Pruebas de Sistema

Estas pruebas se han ido realizando a medida que se desarrollaban los casos de uso. Para realizar estas pruebas se han realizado los pasos definidos en la descripción de los casos usos, primero la secuencia principal y luego los caminos alternativos, y se verifica que el resultado final es como se ha descrito efectivamente verificando que se cumple el requisito funcional.

Pruebas de Requisitos de Gestión de Sesión del Usuario

Id y nombre del requisito	Resultado
RF1.1 - Registrar Usuario	OK
RF1.2 - Iniciar Sesión	OK
RF1.3 - Restablecer Contraseña	OK
RF1.4 - Cerrar Sesión	OK
RF1.5 - Borrar Cuenta	OK

Tabla 8.18: Pruebas de Requisitos de Gestión de Sesión del Usuario

Pruebas de Requisitos de Administración de Usuarios

Id y nombre del requisito	Resultado
RF2.1 - Modificar Perfil de Usuario	OK
RF2.2 - Buscar Amigo	OK
RF2.3 - Añadir Amigos por Código Amigo	OK
RF2.4 - Añadir Amigos por Búsqueda	OK
RF2.5 - Eliminar Amigos	OK
RF2.6 - Modificar Alias a Amigo	OK
RF2.7 - Refrescar Lista de Amigos	OK

Tabla 8.20: Pruebas de Requisitos de Administración de Usuarios

Pruebas de Requisitos de Autómatas como Unidad Abstracta

Id y nombre del requisito	Resultado
RF3.1 - Crear Autómata	OK
RF3.2 - Eliminar Autómata	OK
RF3.3 - Guardar Autómata	OK
RF3.4 - Buscar Autómatas	OK
RF3.5 - Agregar Autómata Buscado	OK
RF3.6 - Hacer público un Autómata	OK
RF3.7 - Renombrar Autómata	OK
RF3.8 - Recargar Lista de Autómatas	OK
RF3.9 - Compartir Autómata con Amigos	OK
RF3.10 - Validar Cadenas de Entrada	OK
RF3.11 - Simular Autómata	OK

Tabla 8.22: Pruebas de Requisitos de Autómatas como Unidad Abstracta

Pruebas de Requisitos de Manipulación del Autómata

Id y nombre del requisito	Resultado
RF4.1 - Añadir Estado	OK
RF4.2 - Borrar Estado	OK
RF4.3 - Añadir Transición	OK
RF4.4 - Borrar Transición	OK
RF4.5 - Editar Nombre de Estado	OK
RF4.6 - Marcar o Desmarcar Estado como Final	OK

Tabla 8.24: Pruebas de Requisitos de Manipulación del Autómata

8.3. Pruebas de Aceptación

Se han realizado dos pruebas de aceptación y en función de la retroalimentación de recibida se ha modificado la interfaz de usuario y la aplicación hasta tener la versión de la aplicación actual.

En esta sección se recoge la retroalimentación recibida en dichas pruebas. Como recordatorio la Primera Prueba se ha realizado cuando la interfaz de usuario era la de Boceto 1, mientras que la Segunda Prueba se ha realizado con la interfaz de usuario Boceto 2.

La retroalimentación de la Prueba 1 puede resumirse en los siguientes puntos:

Retroalimentación	Solución
La página inicial (vista1) es confusa, y cuesta orientarse en ella.	Rediseño de la página principal.
Poco espacio para las simulaciones y validaciones. El autómata ocupa mucho sitio	Eliminación del autómata del simulador, ese espacio esta ahora para validaciones/descripciones
El código de amigo para añadir amigos es engorroso de usar	Implementación de un sistema para buscar amigos públicos
Es difícil encontrar un amigo en especifico si hay muchos	Implementación de un sistema para buscar amigos propios
Extra	En base al trabajo de las dos retroalimentaciones anteriores: implementar un sistema para poder buscar autómatas.

Tabla 8.26: Prueba de Aceptación Uno

La segunda prueba de aceptación se realiza sobre la interfaz gráfica mostrada en Boceto 2 pero de la misma manera, dejando que distintos usuarios potenciales prueben la aplicación, de esta prueba se ha recibido mucho menos retroalimentación sobre problemas o cosas que no les gusten a los usuarios de la aplicación y más sobre posibles adiciones.

Estas adiciones se han anotado como posible trabajo a futuro, aun así se han realizado ligeros cambios a la interfaz de usuario para mejorarla (reflejados en Boceto 3).

Capítulo 9

Conclusiones

9.1. Objetivos Conseguidos

Se ha realizado una aplicación que permite crear y editar autómatas (DFA, NFA y PDA) libremente así como simularlos paso a paso y compartirlos con otros usuarios, el cuál era el objetivo y propósito principal de este TFG.

Se han cumplido todos los objetivos planteados inicialmente para este TFG. Se pretendía desarrollar una aplicación web fácil de usar y aprender que permitiese crear, compartir y simular autómatas (DFA, NFA y PDA).

Mediante el riguroso cumplimiento de todos los casos de uso (verificado mediante Pruebas del Sistema) se verifica que se han cumplido los requisitos funcionales de la aplicación.

Esto es indicativo de que el objetivo del desarrollo de una aplicación web fácil de usar y aprender que permitiese crear, compartir y simular autómatas (DFA, NFA y PDA) se han cumplido.

También se ha cumplido el objetivo de auto-aprendizaje por parte del alumno mediante el desarrollo de esta aplicación se han cubierto en detalle en Resultados de Aprendizaje. Estos resultados se comentan más adelante en detalle en resultados de aprendizaje.

9.2. Objetivos no Conseguidos

Algo no se ha cumplido, son los plazos del desarrollo de la aplicación. Esto no es estrictamente un objetivo pues el objetivo como tal era el desarrollo de la aplicación. Sin embargo, esto es igualmente importante desde el punto de vista del alumno y por eso resulta conveniente tratarlo.

Analizando, el motivo por el que no se ha cumplido ha sido por infravalorar la dificultad de las tareas. Esta infravaloración se ha debido a que no se comprendía completamente lo que había que hacer, llevando a una estimación errónea.

Tomando la realización del TFG como un proyecto de aprendizaje, se han buscado soluciones para que esto no suceda en un futuro cuando el alumno se encuentre trabajando en posiciones de responsabilidad donde el coste de estos retrasos no sea solo el tiempo del alumno si no también el dinero de la empresa:

1. Un estudio minucioso de cada una de las tareas a realizar para poder estimar mejor la cantidad de esfuerzo necesario
2. Si fuera posible consultar a expertos o gente con más conocimientos que el alumno para que revisaran la planificación una vez realizada para verificar las estimaciones.
3. Dejar un colchón de tiempo extra para incidencias no planeadas.
4. En caso de que dichos incumplimientos empiecen a suceder con frecuencia, a pesar de los otros puntos: reservar un hueco y volver a calendarizar la planificación para obtener mejores estimaciones.

9.3. Resultados de Aprendizaje

En esta sección se detallan los resultados del auto-aprendizaje realizado por el alumno mediante la realización del TFG:

- Se ha aprendido a manejar las herramientas y librerías necesarias para desarrollar un aplicación web.
- Se ha aprendido a trabajar por 'libre' teniendo unos objetivos y un tiempo determinado para conseguirlo.
- Se ha aprendido a planificar el desarrollo de un proyecto de software de principio a fin.
- Se ha aprendido a diseñar una interfaz de usuario por propia cuenta y adaptarla en base a la retroalimentación recibida por parte de usuarios
- Se ha conseguido realizar una aplicación simple de utilizar y de aprender por usuarios que conocen algo de teoría de autómatas.
- Se ha aprendido mucho sobre los autómatas finitos y de pila, ya sea tanto sobre su implementación tanto de manera más teórica.

9.4. Trabajo futuro

Hay múltiples aspectos de la aplicación que se podrían mejorar y otras funcionalidades que se podrían incluir. Sin embargo, hacer esto llevaría más tiempo, lo que conllevaría un desarrollo más largo de la aplicación. En cualquier caso es interesante revisarlas ya que son posible direcciones en las que mejorar la aplicación en un futuro:

1. Mejora del sistema de búsqueda para que acepte expresiones regulares, y para que busque en lugar de la cadena exacta de caracteres introducidos los nombres.
2. Mejora del editor de autómatas para que este acepte expresiones regulares como símbolos para las transiciones (de forma que, por ejemplo, una transición '[A-Z]' consumiese cualquier letra mayúscula). Esto reduciría el número de transiciones que es necesario crear para ciertos autómatas.
3. Permitir que se puedan editar transiciones una vez creadas, en lugar de ser necesario borrarlas y volverlas a crear.
4. Expandir las acciones que pueden realizar los amigos, incluyendo enviarse mensajes entre ellos o bloquear a otros usuarios.
5. Hacer que la aplicación se vea correctamente sin que la ventana tenga que funcionar a tamaño completo.

Parte IV

Apéndices

Apéndice A

Manual de Instalación

La aplicación se ha desarrollado para ejecutarla en una maquina virtual con las siguientes características. Se recomienda usar una maquina virtual con características similares para la instalación.

Distributor ID	Ubuntu
Description	Ubuntu 20.04.4 LTS
Release	20.04
Versión de Linux	Linux virtual 5.4.0-107-generic #121-Ubuntu SMP Thu Mar 24 16:04:27 UTC 2022 x86_64 x86_64 x86_64 GNU/Linux
MemTotal	2 GB (2030636 kB)
MemDisco	8 GB (8589934.592 kB)

Tabla A.2: Configuración Máquina Virtual

El código de la aplicación se encuentra alojado en un repositorio en GitLab que puede ser accedido desde la siguiente url: `https://gitlab.inf.uva.es/adrseba/TFG-INF`

Hay 4 carpetas importantes aquí, backend, public, src y sripts. La carpeta backend contiene el código para crear el servidor así como la base de datos, public contiene recursos necesarios para el funcionamiento de src, src contiene el código del frontend mientras que script contiene dos scripts en bash: uno para ejecutar la aplicación (arranca) y otro para detenerla (para).

En caso de querer instalar la aplicación es necesario realizar ciertos cambios, crear una base de datos para el backend, crear una cuenta en OKTA y configurar la aplicación y modificar ciertas partes del código. Se procede a explicar estos en detalle a continuación

Base de Datos

Dentro de backend está el archivo *backend.sql* a partir del cuál se pueden crear las tablas de la base de datos necesaria para la aplicación.

La base de datos debe ser PostgreSQL y tener la siguiente configuración:

host	localhost
port	5432
database	backend
user	adrian
password	asc2mil21.

Tabla A.4: Configuración de la Base de Datos

Donde host es el nombre del servidor, port el número de puerto del servidor, database el nombre de la base de datos, user el nombre del usuario que accederá a la base de datos y realizará las operaciones y password su contraseña.

Esta configuración puede ser distinta siempre que se modifique en el archivo *dbUtils.js* la constante *cn* la cuál describe la configuración de la base de datos a conectar.

OKTA

La aplicación utiliza Okta como proveedor de identidad. Es necesario crear una cuenta como desarrollador en OKTA (en caso de que no se tenga una) y crear una aplicación de integración para esta aplicación.

Se utiliza *OIDC - OpenID Connect como Sign-in* method y *Single-Page Application* para *Application Type*. Tras esto se debe marcar todos *Grant type* posibles. En cuanto a *Assignments* si no hay más aplicaciones registradas se puede marcar *Allow everyone in your organization to access*, si las hubiera sería necesario seleccionar *Limit access to selected groups* (o *Skip group assignment for now* aunque habría que configurarlo más tarde) y configurar el registro de usuarios para que cuando un usuario se registre en la aplicación sea asignado a dicho grupo. *Sign-in redirect URIs* debe ser la dirección desde la cuál la aplicación llama a OKTA, *Sign-out redirect URIs* debe ser la misma que *Sign-in redirect URIs*.

Una vez la aplicación este creada es necesario abrirla e ir a *Sign On* y en *OpenID Connect ID Token* seleccionar *Okta URL*.

Además, se debe habilitar *Self-service Registration* (*Directory*) con verificación de correo y seleccionar únicamente (desmarcando el resto) *nickName* como *Registration form fields*. *Default redirect* debe ser el mismo que se puso para *Sign-in redirect URIs*. *Add to Sign-In widget* también debe ser marcado y *Assign to group* no importa si previamente se escogió *Allow everyone in your organization to access*, pero si se escogió otra opción aquí debería ir el grupo de usuarios a los que se les permitiría el acceso.

En *Authentication* es posible cambiar las opciones de la password introducida, esto se ha realizado para la aplicación pero no es necesario para que funcione.

Finalmente en *API* (*Security*) es necesario ir a *trusted origins* y añadir como origen la aplicación (misma URI que para *Sign-in redirect URIs*) con permiso para Redirect y CORS.

Esto es lo que hay que hacer para configurar la aplicación de integración de OKTA, además de esto es necesario modificar el código de la aplicación, en *backend/OktaConfig.js*: url debe modificarse para que se ajuste a la cuenta del usuario (modificando el *dev-91268010* por el correspondiente) y

Authorization (dentro de headers) debe cambiarse por *SSWS* +su_token_de_autorización. Donde su_token_de_autorización es un token que se debe crear en *API* (*Security*) en *Tokens*.

Finalmente en *src/redux/config.js* es necesario cambiar *issuer* y *baseUrl* por las correspondientes (modificando solo la parte de dev-91268010) y *clientId* debe modificarse por el que viene en la aplicación de integración que se ha creado.

Otras Modificaciones del Código

En todos los archivos de src/redux/sagas/requests se debe de modificar la constante *direction* por la nueva dirección url y puerto en el que está el servidor.

En backend/server.js es necesario añadir a la constante *corsOptions* en *origin* la url del cliente. Así como modificar las variables *port* y *address* por el puerto y dirección IP local en donde va a correr el servidor.

Tras los Cambios

Una vez los cambios están hechos es necesario compilar la aplicación mediante el comando *npm run build* desde el directorio raíz. Esto generará una carpeta build en el directorio raíz con el código del cliente compilado. Tras esto se debe ejecutar la build mediante:

serve -s -n path_to_build port_number_to_run reemplazando *path_to_build* por el path a la carpeta build y *port_number_to_run* por el puerto en el que se ejecutará el cliente.

El servidor debe ejecutarse mediante *node path_to_server* donde *path_to_server* es el path al fichero *backend/server.js*.

Tanto el servidor como el cliente deben estar ejecutándose al mismo tiempo para que la aplicación funcione. Es también recomendable redirigir la salida tanto de server.js como de build a otro archivo para poder ver los logs.

Para que tanto el cliente como el servidor se ejecuten simultáneamente y se redirija su salida se han utilizado los comandos en bash que se encuentran en scripts, es decir, *arranca.bash* y *para.bash*. Si bien estos no funcionarán correctamente (en el sentido de arrancar y parar la aplicación web) fuera de la máquina virtual del alumno Estos se pueden utilizar como ejemplo de como crear unos ejecutables en bash para lanzar el servidor y el cliente simultáneamente.

Apéndice B

Manual de Usuario

Se puede acceder a la página web a través de la dirección `http://virtual.lab.inf.uva.es:20092/`

B.1. Sesión no Iniciada

B.1.1. Comunidad

Cuando un nuevo usuario accede a la aplicación esta es la primera página que ve. Aquí puede buscar autómatas públicos, también usando los botones de abajo que tiene disponible puede ver (icono del ojo) o simular los autómatas buscados (icono de play)

Ver un autómata lleva a una vista del editor de autómatas (descrito en Sesión Iniciada), pero con todas las opciones de modificar el autómata, excepto visualmente eliminadas. Mientras que Simularlo lleva del Simulador (también descrito en Sesión Iniciada).

Figura B.1: Comunidad sin Iniciar Sesión

B.1.2. Iniciar Sesión/Registarse

En Iniciar Sesión/Registarse se puede iniciar sesión, para ello solo tiene que introducir su correo y contraseña. También puede marcar el botón de 'recordarme' para poder acceder a su cuenta omitiendo estos pasos.

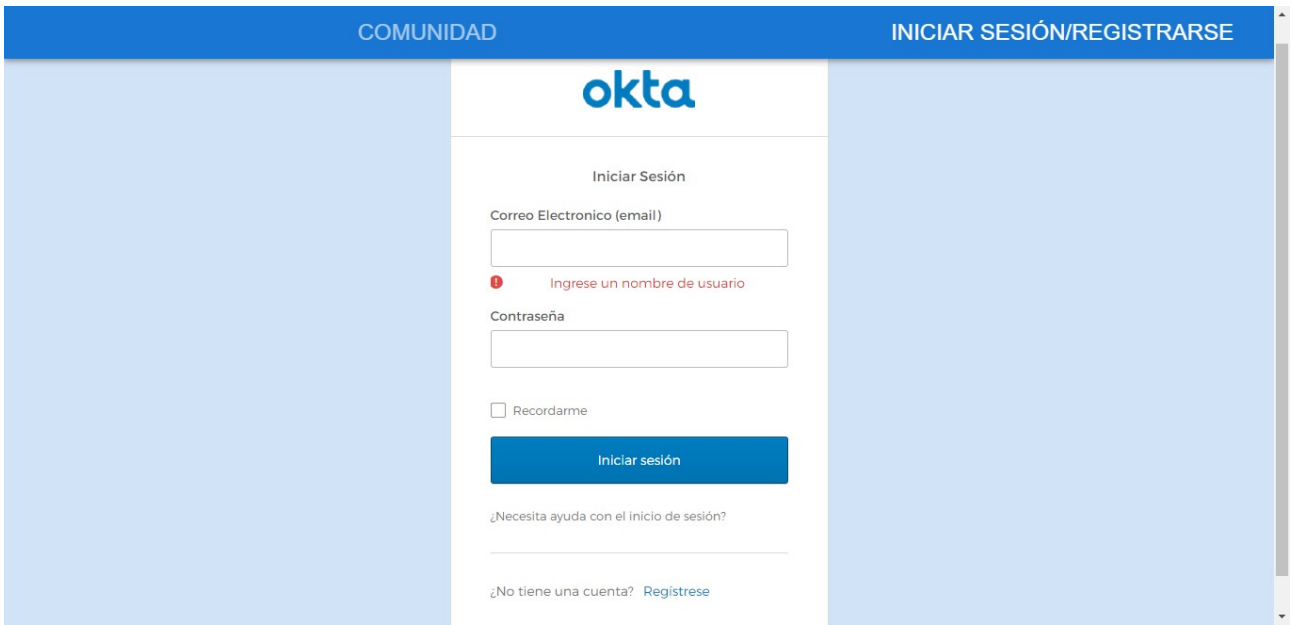

Figura B.2: Iniciar Sesión

En caso de que quiera recuperar la contraseña basta con clickar en pedir ayuda y luego en he olvidado la contraseña, esto redirigirá al usuario a esta página, en la cual tiene que introducir su correo, tras esto le llegará un email a su correo para que introduzca una nueva contraseña.

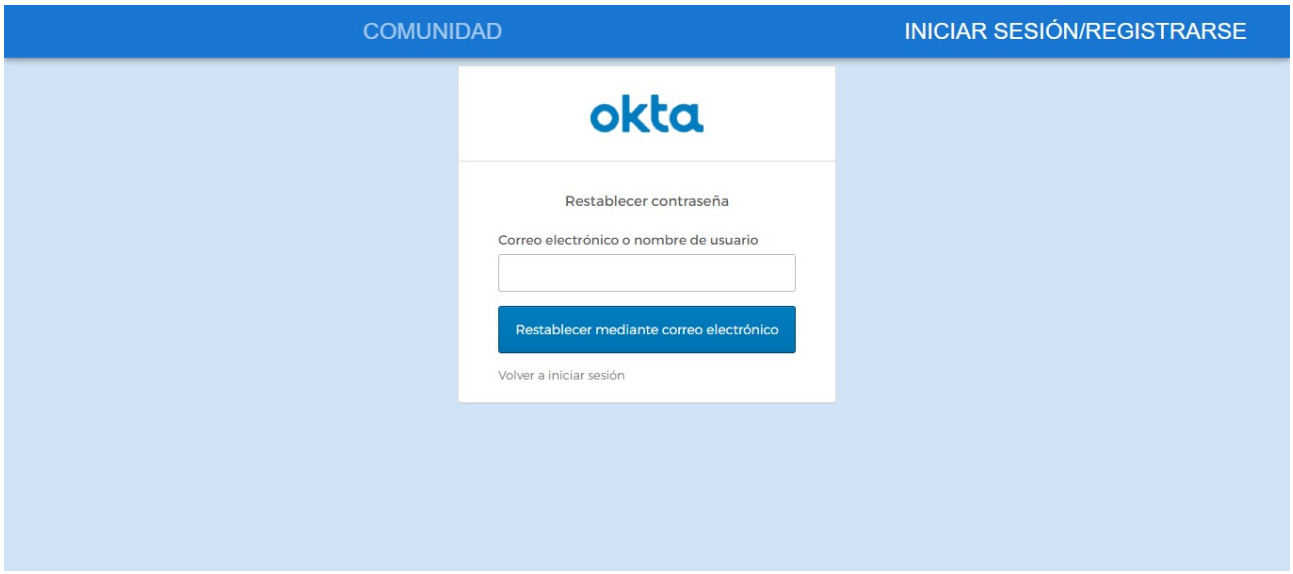

Figura B.3: Recuperar Contraseña

Para registrase desde la página de inicio de sesión se hace click en registrarse lo cuál lleva a la siguiente página. Aquí hay que introducir el correo electrónico, una contraseña y el nombre de usuario dentro de la aplicación. También es necesario confirmar el correo para completar el registro.

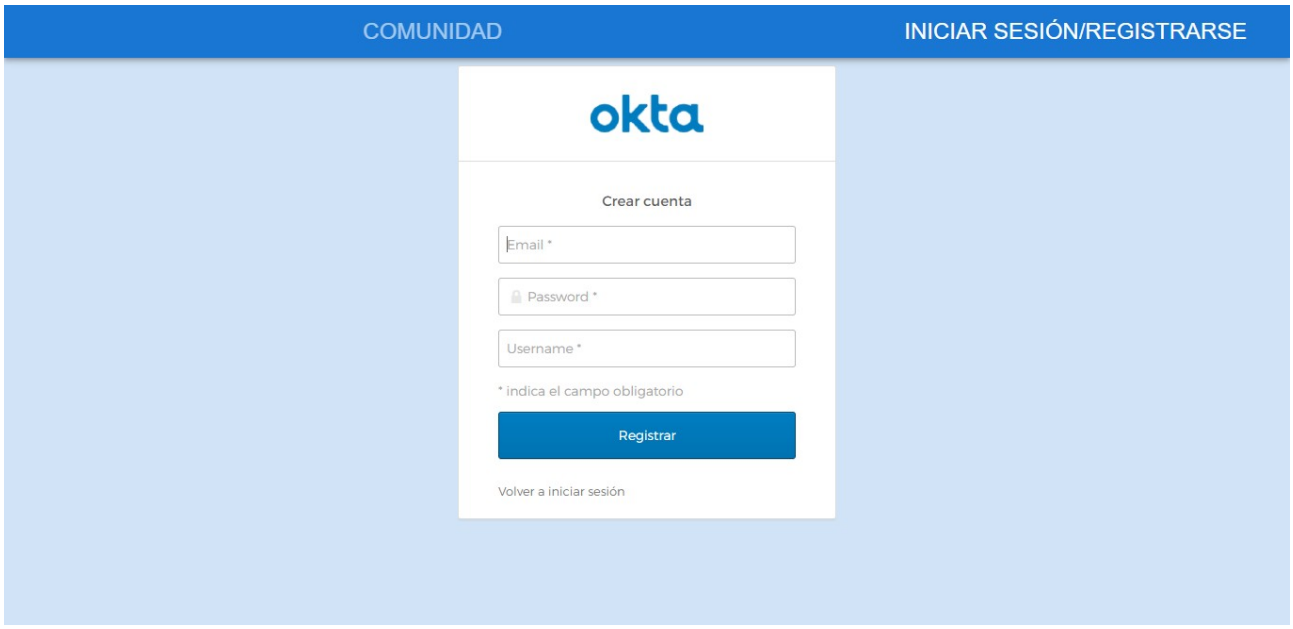

Figura B.4: Registro de Usuario

B.2. Sesión Iniciada

Comunidad

Además de lo que se podía hacer sin tener la sesión iniciada ahora también se puede copiar autómatas de la lista pública a la colección privada del usuario (símbolo '+' más a la izquierda) o tratar de agregar a un usuario como amigo (símbolo de correo a la derecha del '+').

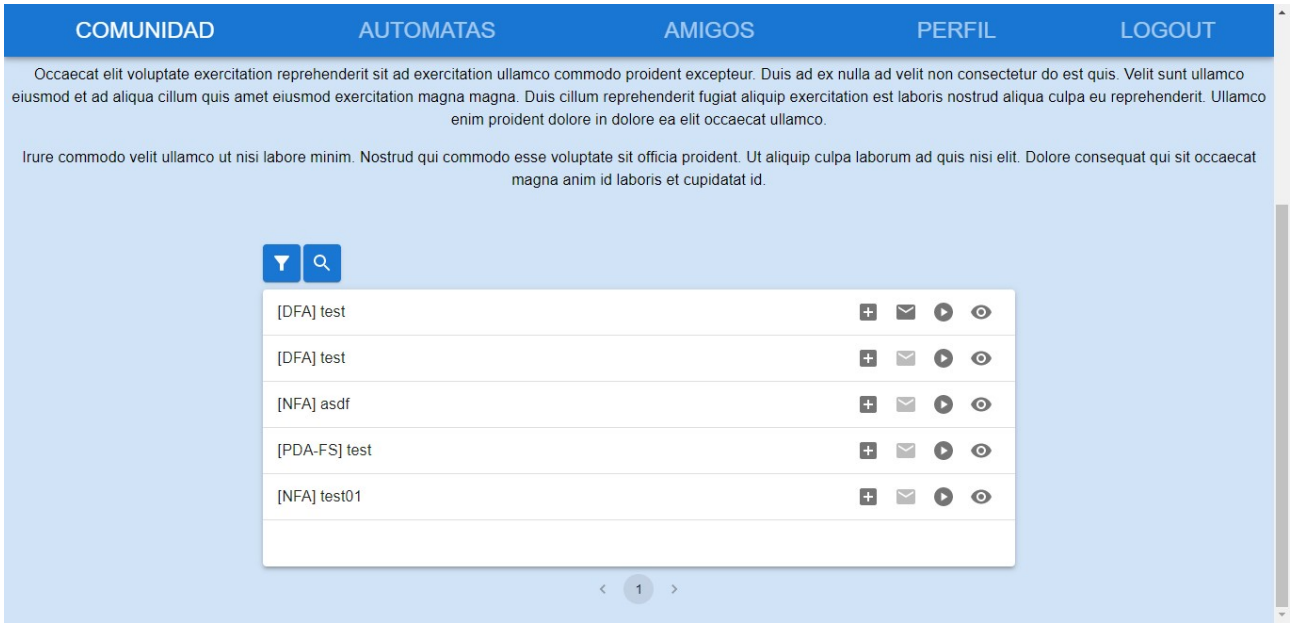

Figura B.5: Comunidad Sesión Iniciada

B.2.1. Autómatas

Colección privada de autómatas del usuario. Aquí un usuario puede realizar múltiples acciones:

Además también se puede recargar la lista clickando en el símbolo recargar en la parte superior y se pueden crear un nuevo autómata presionando el botón '+', introduciendo el nombre y seleccionando el tipo

- Si se hace click en el nombre del autómata se puede ver el autómata en el lado izquierdo tal y como aparece en la imagen.
- Si se hace click en el símbolo de más a la izquierda de un autómata se irá al editor de autómatas para dicho autómata
- Si se hace click en el botón de play, igual que para autómatas públicos se irá a simular el autómata
- Si se hace click en el botón de compartir se pueden elegir amigos aceptados para compartir el autómata como se puede ver en la imagen inferior

- Si se hace click en el lapicero podrá sale una ventana para editar el nombre del autómata.
- Si se hace click en el la bola del mundo se marca o desmarca un autómata como público.
- Si se hace click en la basura se borra el autómata

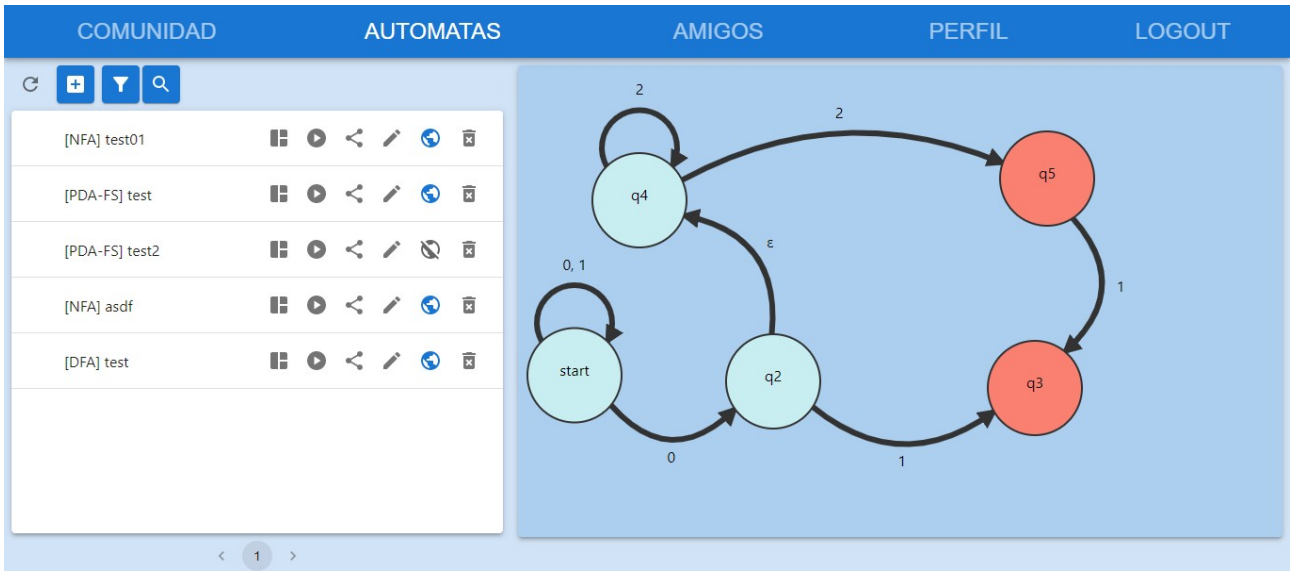

Figura B.6: Autómatas

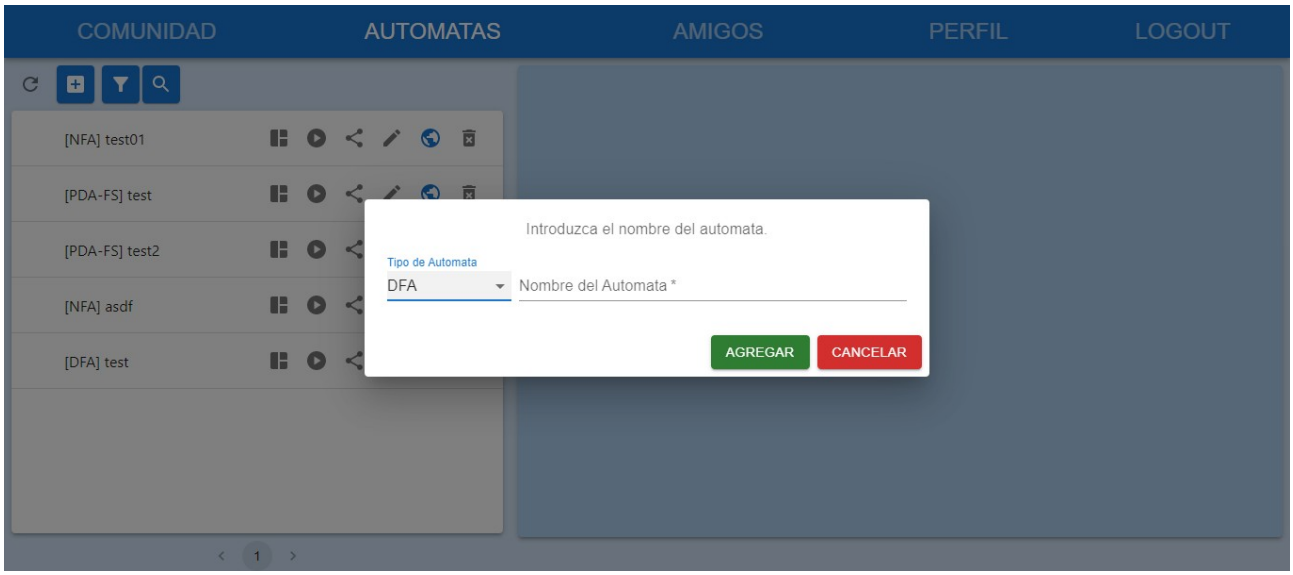

Figura B.7: Crear Autómatas

Así como buscar autómatas de la misma manera que se podía en Comunidad. Los filtros en verde indican que el autómata debe ser de ese tipo mientras que en rojo que no debe ser de ese tipo.

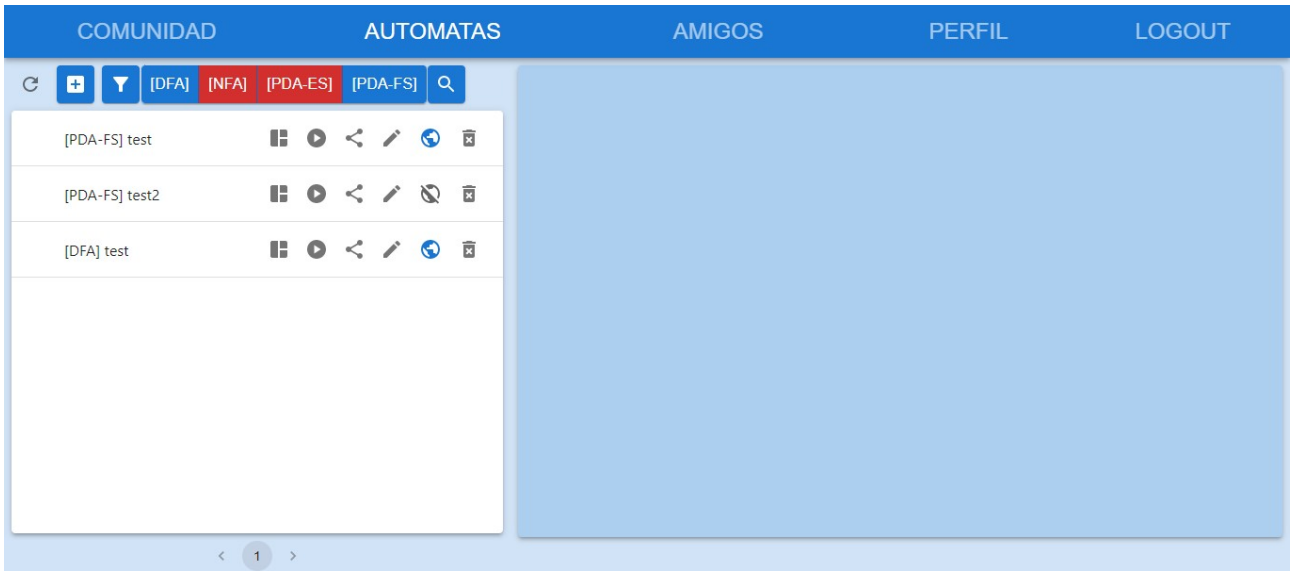

Figura B.8: Autómatas Filtrados

B.2.2. Amigos

En amigos hay dos filtros, propios y públicos (uno siempre en rojo y otro siempre en verde). En esta imagen se muestran los propios.

El color de los amigos indica el estado, verde indica que se ha confirmado y se puede compartir un autómatas con ellos, mientras amarillo y gris indica que se está a la espera de confirmación (con amarillo indicando que debe confirmar y gris que debe esperar a que le confirmen).

Para confirmar se hace click en el símbolo □, para editar el nombre en el símbolo del lapicero y para borrarlo en la 'x'.

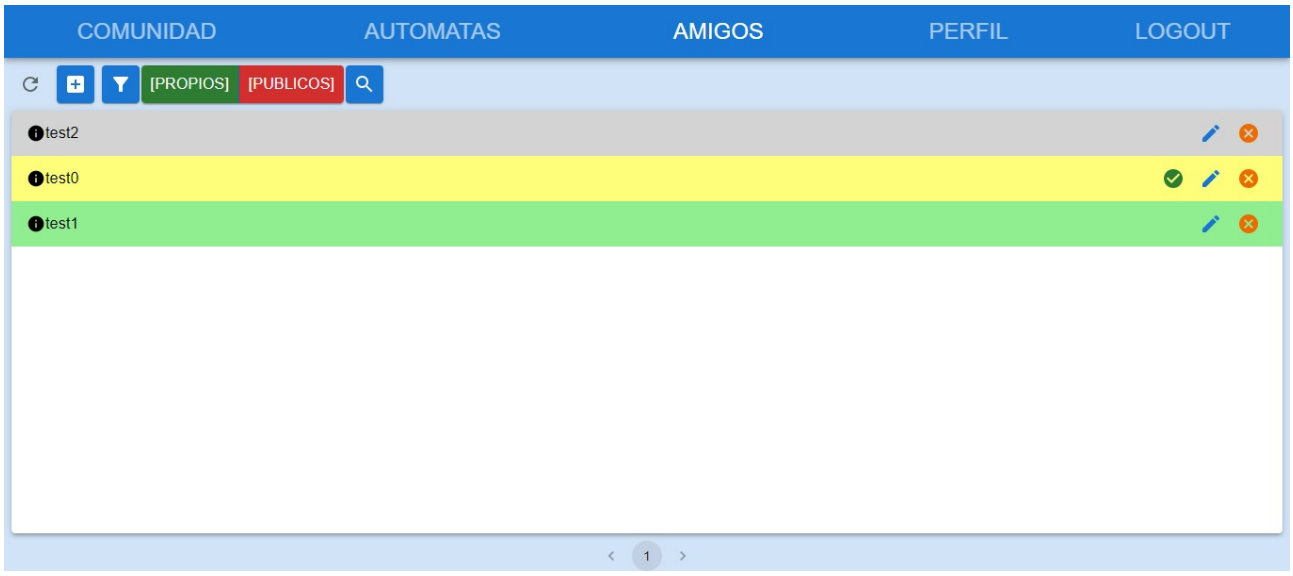

Figura B.9: Amigos Propios

De forma similar a autómatas se puede recargar la lista clickando en refresh y en el símbolo '+'

se puede crear un nuevo amigo.

Para crear un nuevo amigo es necesario introducir el código de amigo y el nombre que se le va a asignar (nombre que aparecerá en la lista de amigos).

Si el campo de nombre se deja en blanco se cogerá el nombre actual del usuario a agregar (como nota el usuario agregado aparecerá en gris en la lista pues se debe esperar su confirmación).

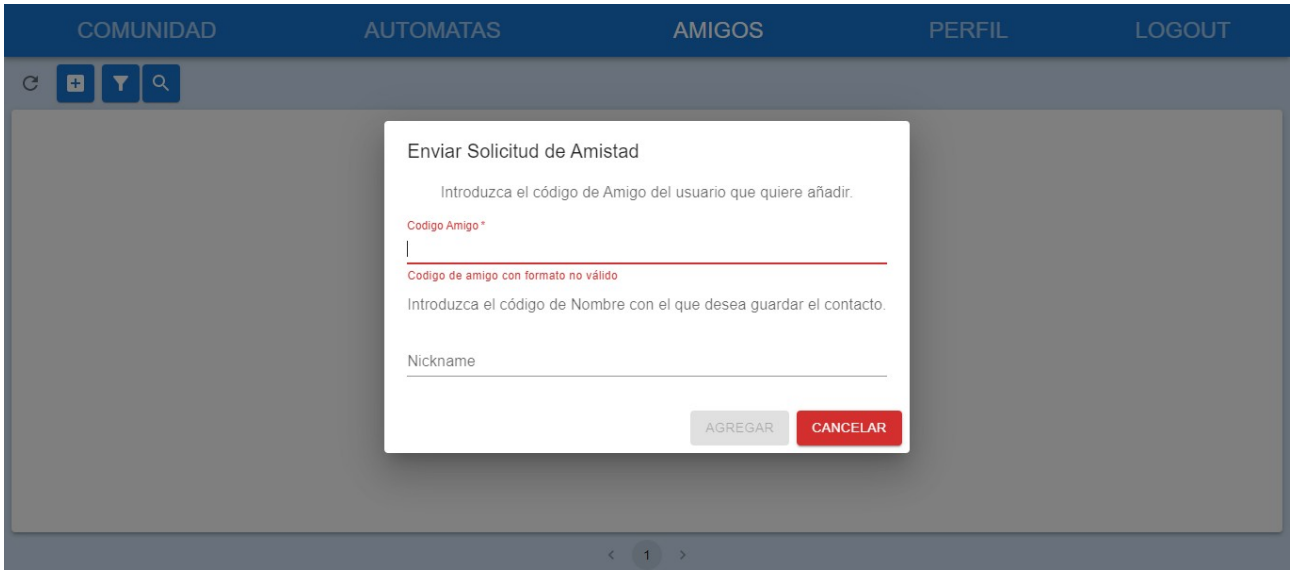

Figura B.10: Diagrama de Casos de Uso de Gestión de Sesión del Usuario

Cuando se filtra por 'públicos' es algo distinto. En este caso aparece la lista de usuarios públicos, la única interacción posible con estos es clickar en el icono del correo, lo que causará que se trate de agregar a dicho usuario como amigo (en propios aparecerá dicho usuario en gris mientras que en la lista del usuario que se trata de agregar aparecerá en amarillo).

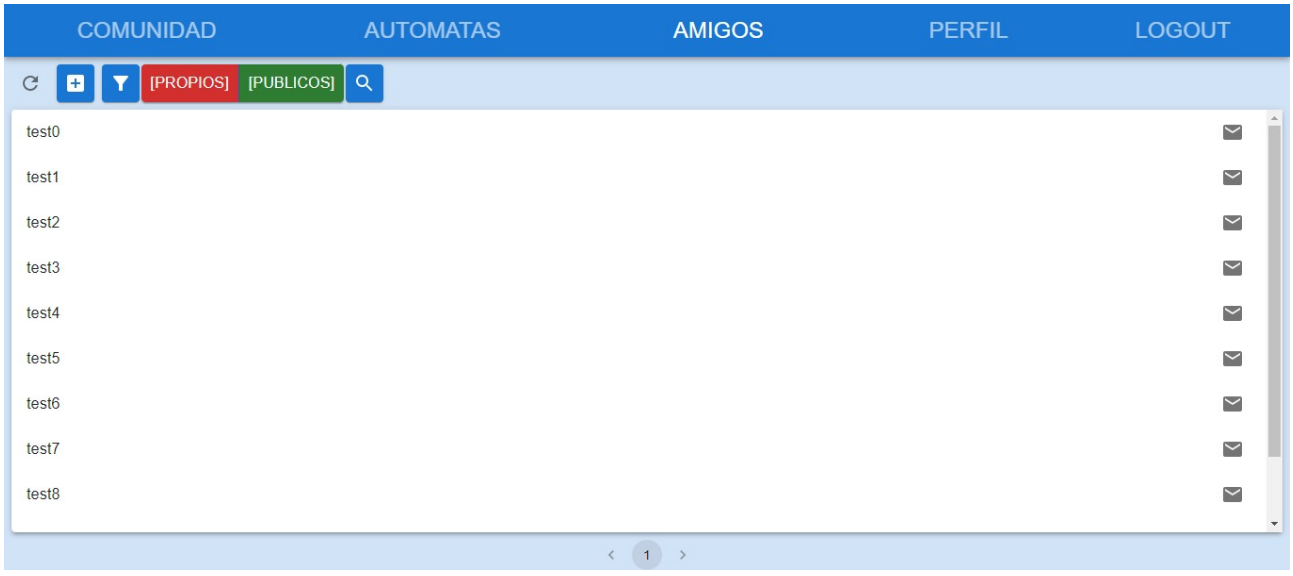

Figura B.11: Usuarios Públicos

B.2.3. Perfil

En Perfil se puede editar directamente el nombre del usuario (basta con clickar sobre él y ponerse a editarlo), copiar o generar un nuevo código de amigo (clickar los botones a la izquierda del código de amigo) o modificar los otros ajustes existentes.

El resto de ajustes son checkboxes que se pueden marcar o desmarcar o listas con valores prefijados.

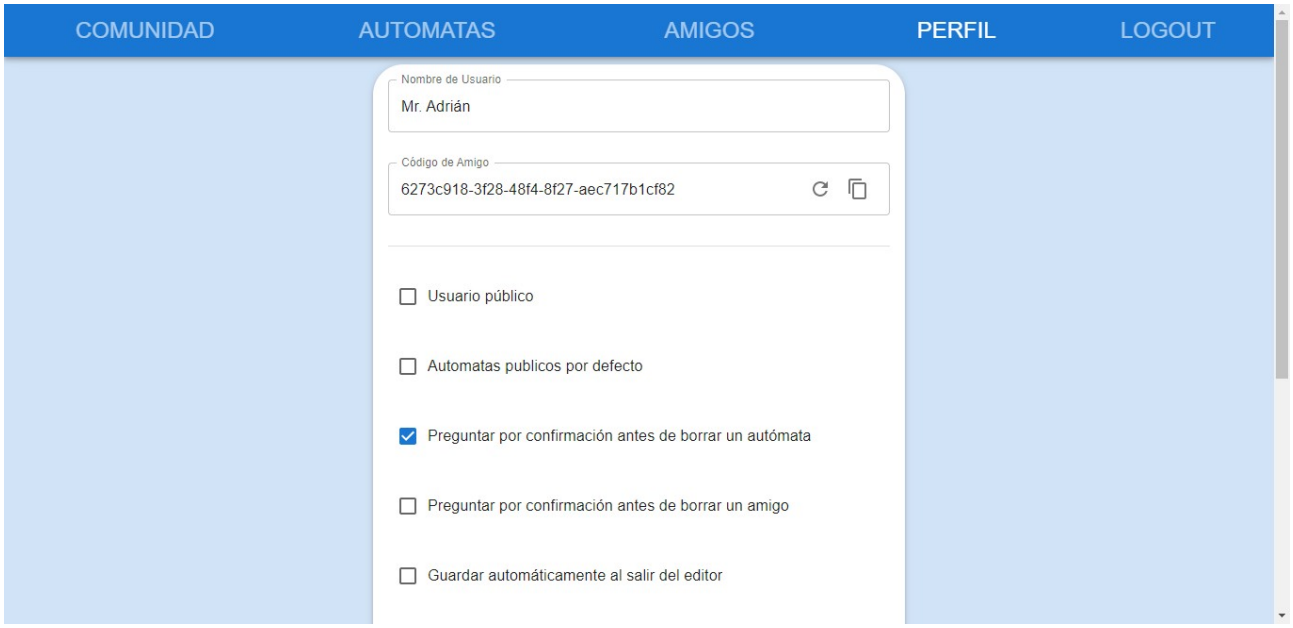

Figura B.12: Perfil de Usuario 1

Para que se guarden los cambios es necesario hacer click en el botón 'Guardar Cambios' al fondo de la página, justo encima de 'Borrar Cuenta', si no ningún cambio será guardado.

Presionar Borrar Cuenta permitirá al usuario borrar permanentemente su cuanta (tras confirmar que realmente quiere hacerlo).

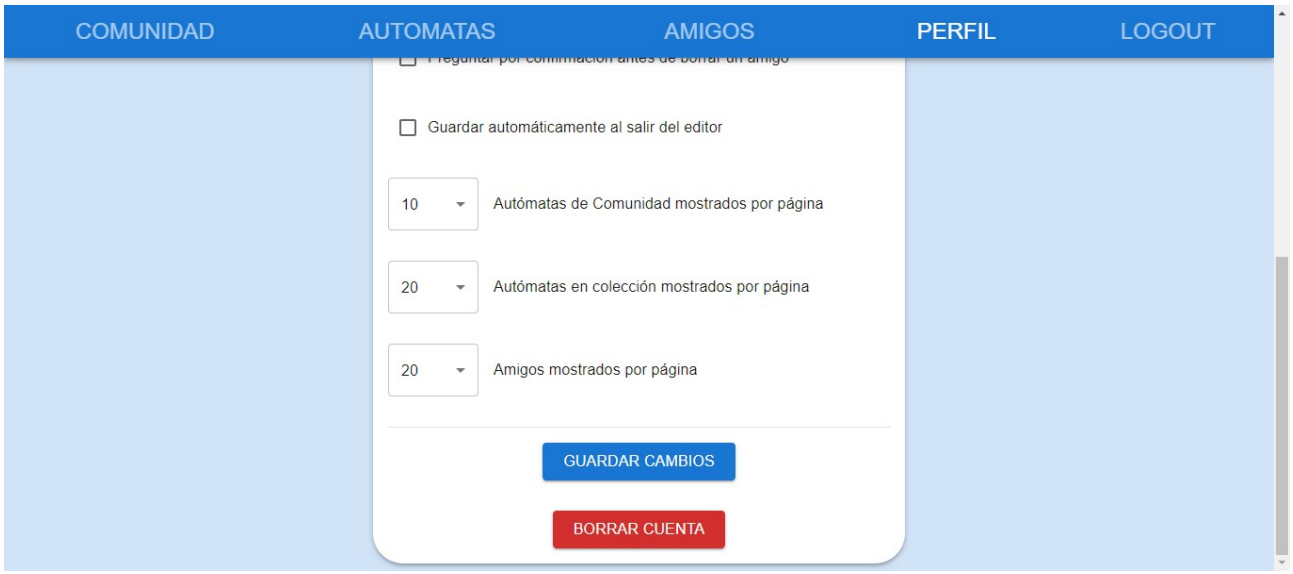

Figura B.13: Perfil de Usuario 2

B.2.4. Logout

Hacer click en Logout Cierra la sesión del usuario y lo redirige a Comunidad.

B.2.5. Editor de Autómatas

El editor de autómatas es posible guardar los cambios (clickando en 'Guardar Cambios'), ver todos los controles (clickando en 'Ayuda') y pasar al simulador de autómatas ('Simulador de Autómatas').

Controles:

- Crear Estado: Shift + click izquierdo sobre el fondo
- Crear Transición: Shift + click izquierdo en nodo y arrastrar hasta nodo
- Editar nombre de estado: doble click sobre un nodo
- Marcar o desmarcar estado como final: click derecho sobre un nodo
- Seleccionar Elemento: click izquierdo sobre nodo o transición
- Borrar Elemento Seleccionado: Backspace Suprimir(Supr)
- Mover nodo: click izquierdo sobre un nodo y arrastrar
- Mover fondo: click izquierdo sobre el fondo y arrastrar
- Zoom: ruleta del ratón

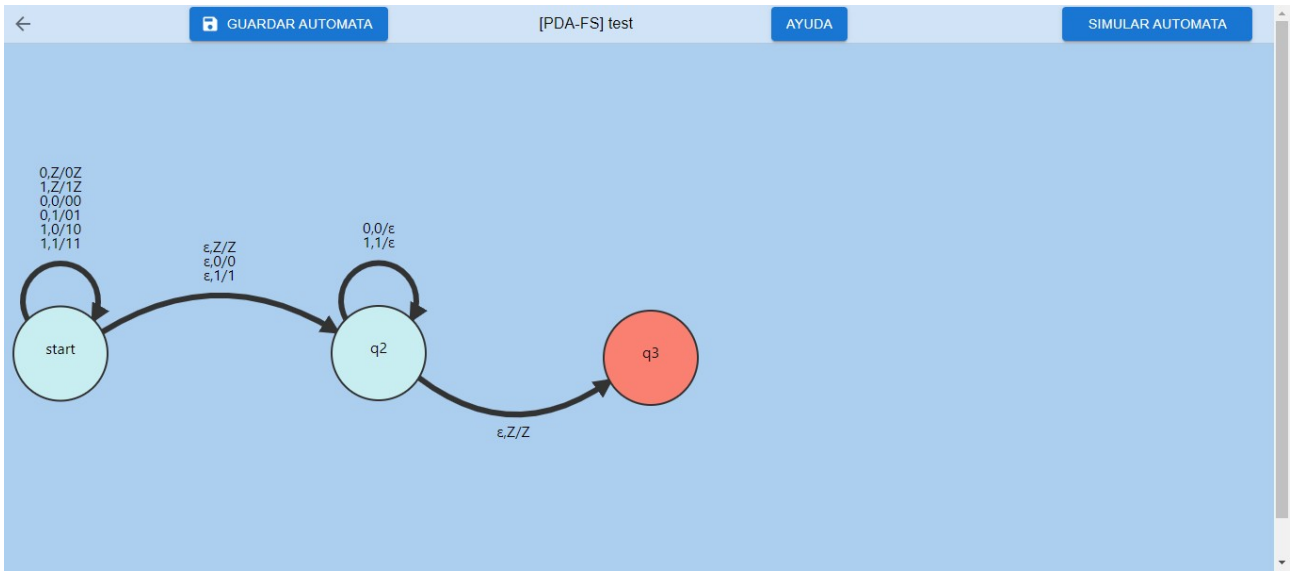

Figura B.14: Editor de Autómatas

Cuando se trata de crear una nueva transición se muestra un ventana para que el usuario introduzca los caracteres para la transición.

Para los DFA se debe meter un carácter, mientras que los NFA se puede dejar vacío y se interpretará como una cadena vacía

Figura B.15: Ventana de Transición para Autómata DFA

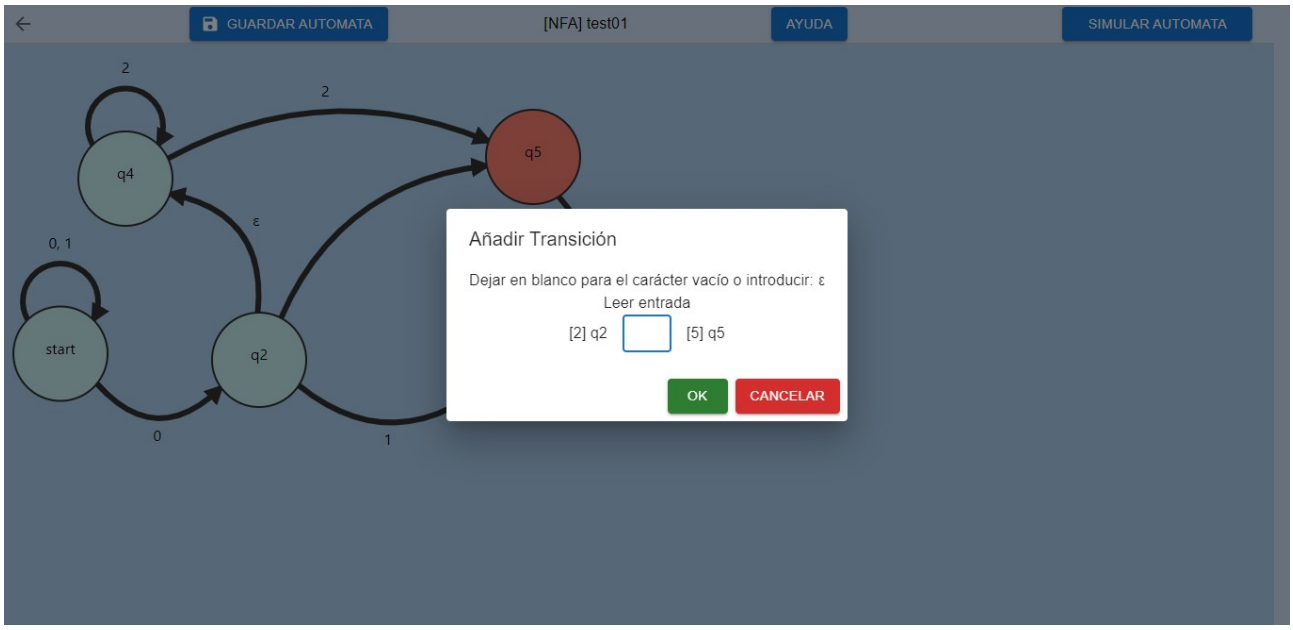

Figura B.16: Ventana de Transición para Autómata NFA

Por otro lado los PDA tienen 3 sitios para introducir caracteres. El primero se corresponde al carácter a consumir (dejar en blanco para cadena vacía), en la segunda para el símbolo que debe haber en la cima de la pila para que se realice la transición (no puede dejarse vacío ni ser la cadena vacía), mientras que el último campo se corresponde a los símbolos (pueden ser más de uno o ninguno) por los que reemplazar la cima.

Nota la pila va de izquierda a derecha. Ejemplo: si la pila es ’ZZZ’ y se debe reemplazar la pila por ’ab’, entonces la pila resultante será ’abZZZ’.

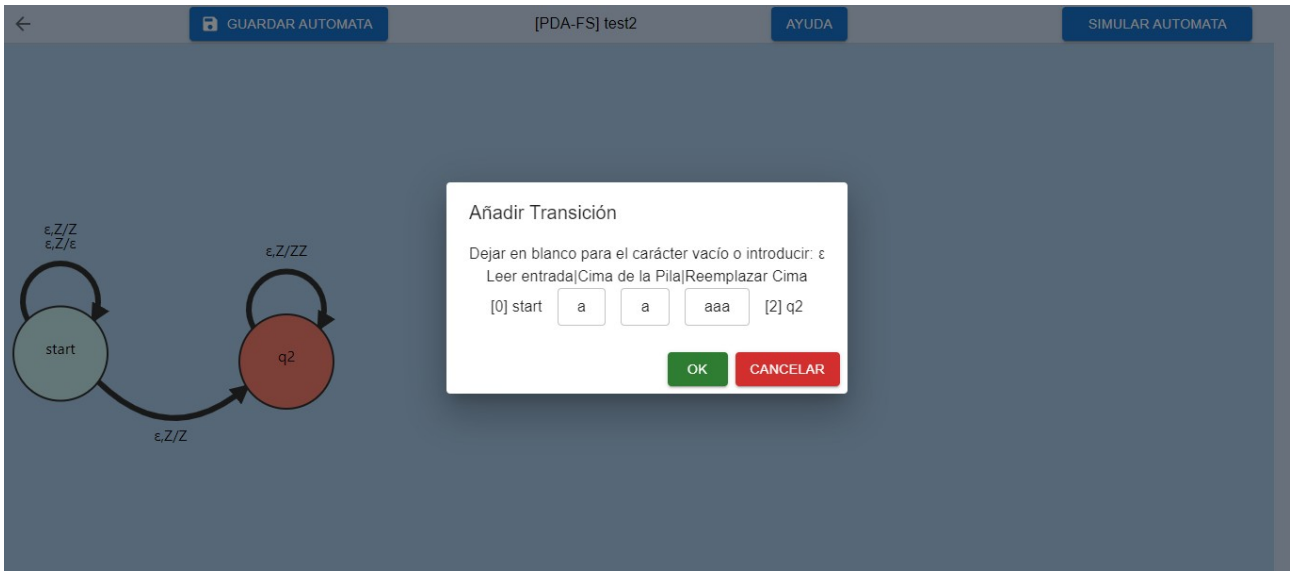

Figura B.17: Ventana de Transición para Autómata PDA

B.2.6. Simulador

En el Simulador se pueden validar múltiples cadenas de entrada o validar una en detalle (se muestra cada descripción instantánea generada).

Para validar múltiples cadenas de entrada se introducen dichas cadenas en la caja de texto 'Inputs', separadas por un salto de línea y luego se presiona el botón 'Validar'.

Si se quiere validar una cadena en detalle, esta se introduce en la caja de texto inferior, 'Simulación Paso a Paso', y se presiona 'Iniciar Simulación'

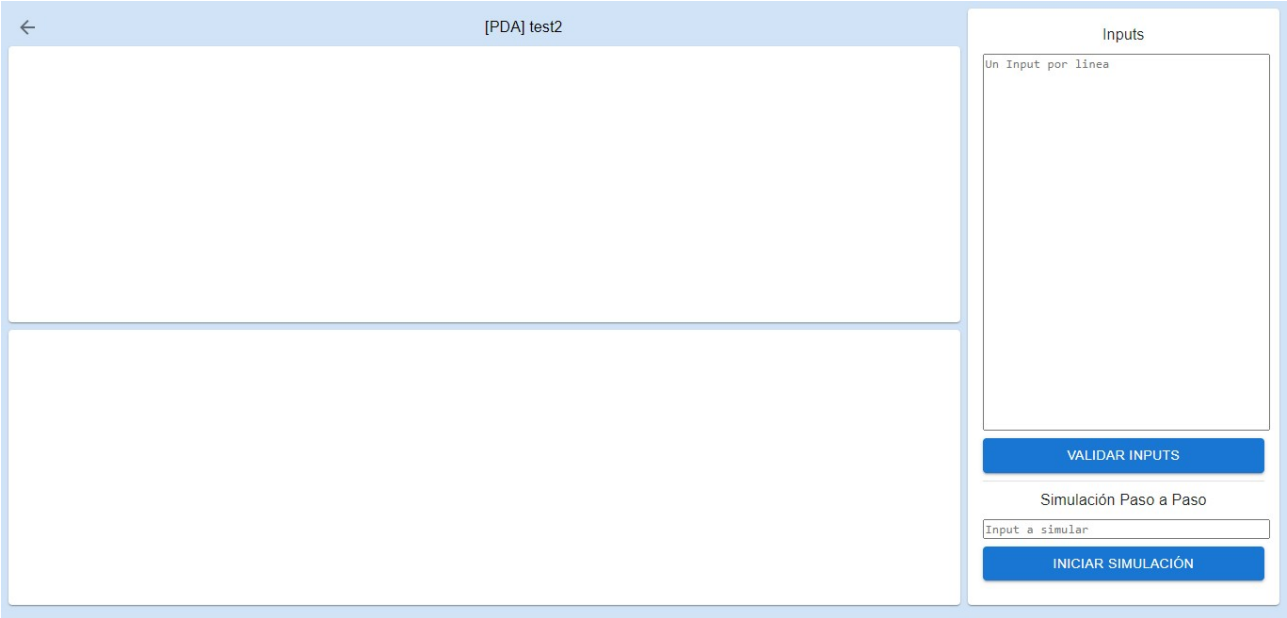

Figura B.18: Simulador

Si se hace lo anterior el resultado es la siguiente imagen.

En la caja de superior de la derecha se encuentran las descripciones instantáneas, en verde indica que está aceptada, en rojo que está rechaza, y en gris que no esta en ninguno de los estados anteriores. Por otro lado la intensidad del color determina (verde oscuro, rojo oscuro, gris oscuro) si son las últimas transiciones generadas o correspondientes a un paso anterior.

Para avanzar un paso se click el botón de play a la izquierda del botón rojo de stop, mientras que para parar la simulación se presiona el de stop. La simulación paso a paso termina automáticamente si se encuentra una descripción instantánea aceptada.

Por otro lado en la ventana inferior se encuentran las cadenas de la caja de 'Inputs' y su estado, aceptas o rechazadas

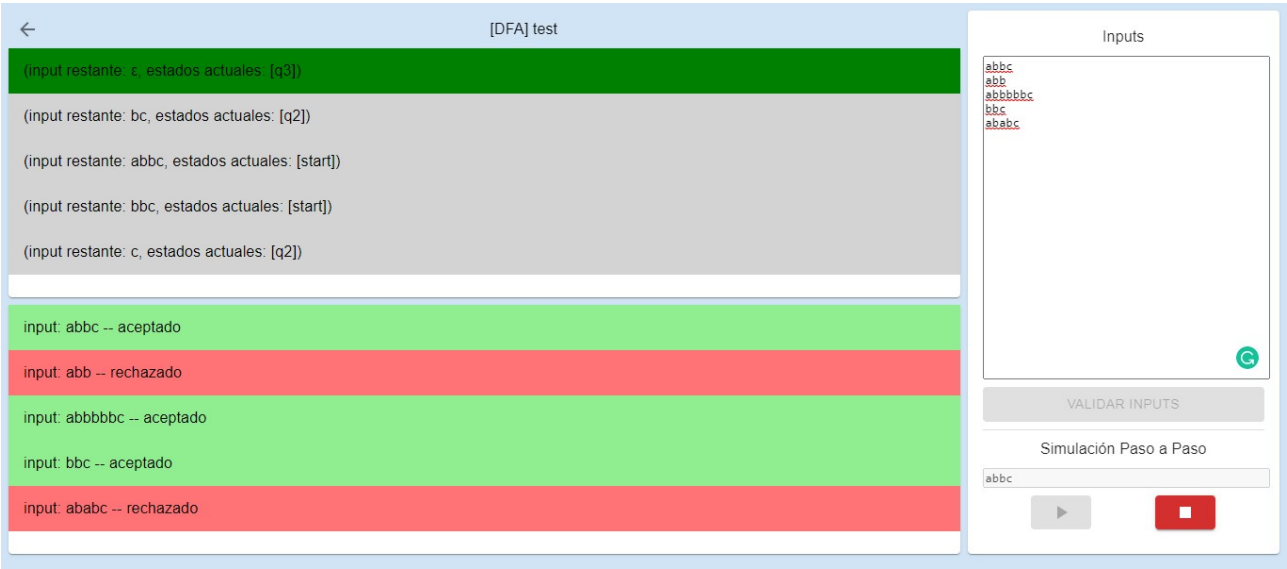

Figura B.19: Simulador con Validaciones

Bibliografía

[1] *Agile Methodology*. url: https://www.digite.com/agile/agile-methodology/. (last accessed: 2022-03-10).

[2] *Okta*. url: https://www.okta.com/. (last accessed: 2022-03-10).

[3] *d3*. url: https://d3js.org/. (last accessed: 2022-03-10).

[4] *d3v3 documentation*. url: https://devdocs.io/d3~3/. (last accessed: 2022-03-10).

[5] *react*. url: https://reactjs.org/. (last accessed: 2022-03-10).

[6] *react-redux*. url: https://react-redux.js.org/. (last accessed: 2022-03-10).

[7] *redux-saga*. url: https://redux-saga.js.org/. (last accessed: 2022-03-10).

[8] *materila-ui*. url: https://mui.com/. (last accessed: 2022-03-10).

[9] *pg-promise*. url: http://vitaly-t.github.io/pg-promise/index.html. (last accessed: 2022-03-10).

[10] *Okta sign-in widget*. url: https://developer.okta.com/docs/guides/embedded-siw/main/. (last accessed: 2022-03-10).

[11] *Salario ingeniero informático*. url: https://es.talent.com/salary?job=ingeniero+informatico. (last accessed: 2022-03-10).

[12] R. Motwani y J. Ullman J. Hopcroft. *Introducción a la teoría de autómatas, lenguajes y computación*. Pearson Education, Inc, 2007. isbn: 9788478290888.

[13] Wikipedia. *Construcción de Subconjuntos*. url: https://es.wikipedia.org/wiki/Construcci%5C%C3%5C%B3n_de_subconjuntos. (last accessed: 2022-03-10).

[14] Kyle Dickerson. *automatonsimulator*. url: https://automatonsimulator.com/. (last accessed: 2022-03-10).

[15] caleb531. *automata*. url: https://github.com/caleb531/automata. (last accessed: 2022-03-10).

[16] Alessio Cecconi. *PySimpleAutomata*. url: https://pysimpleautomata.readthedocs.io/en/latest/index.html. (last accessed: 2022-03-10).

[17] Susan H. Rodger. *JFLAP*. url: https://www.jflap.org/. (last accessed: 2022-03-10).

[18] SachinKumar105. *Converting-a-NFA-to-DFA*. url: https://github.com/SachinKumar105/Converting-a-NFA-to-DFA. (last accessed: 2022-03-10).

[19] Wikipedia. *Modelo de Dominio*. url: https://es.wikipedia.org/wiki/Modelo_de_dominio. (last accessed: 2022-03-10).

[20] *Unicidad del id de OKTA*. url: https://support.okta.com/help/s/article/Will-the-Okta-user-id-be-unique-across-all-Okta-instances?language=en_US#:~:text=Yes.,unique%5C%20across%5C%20all%5C%20Okta%5C%20tenants.. (last accessed: 2022-03-10).

[21] Wikipedia. *Patrones de Diseño*. url: https://es.wikipedia.org/wiki/Patr%5C%C3%5C%B3n_de_dise%5C%C3%5C%B1o. (last accessed: 2022-03-10).

[22] Excalidraw. *Excalidraw*. url: https://excalidraw.com/. (last accessed: 2022-03-10).

[23] Microsoft. *Paint*. url: https://en.wikipedia.org/wiki/Microsoft_Paint. (last accessed: 2022-03-10).

[24] w3schools. *Jquery*. url: https://www.w3schools.com/jquery/jquery_intro.asp. (last accessed: 2022-03-10).

[25] Wikipedia. *Angular*. url: https://en.wikipedia.org/wiki/Angular_(web_framework). (last accessed: 2022-03-10).

[26] Wikipedia. *Vue*. url: https://en.wikipedia.org/wiki/Vue.js. (last accessed: 2022-03-10).

[27] Nacho Blanco. *patron-angular*. url: https://openwebinars.net/blog/que-patron-usa-angular-mvc-o-mvvm/. (last accessed: 2022-03-10).

[28] Issam Nacim. *redux-saga flow chart*. url: https://medium.com/@nacim.issam/redux-saga-a4e9fb83fa4d. (last accessed: 2022-03-10).

[29] Wikipedia. *Tests de Unicidad*. url: https://en.wikipedia.org/wiki/Unit_testing. (last accessed: 2022-03-10).

[30] Wikipedia. *Tests de Integración*. url: https://en.wikipedia.org/wiki/Integration_testing. (last accessed: 2022-03-10).

[31] Wikipedia. *Tests de Sistema*. url: https://en.wikipedia.org/wiki/System_testing. (last accessed: 2022-03-10).

[32] Wikipedia. *Tests de Aceptación*. url: https://en.wikipedia.org/wiki/Acceptance_testing. (last accessed: 2022-03-10).

[33] emailfake. *Generador de Dummy Emails*. url: https://emailfake.com/. (last accessed: 2022-03-10).