Pontificia Universidad Javeriana Cali
Facultad de Ingeniería.
Maestría en Ingeniería de Software
Proyecto de Grado.

Aplicar el agente inteligente basado en aprendizaje por refuerzo RELOAD para realizar pruebas de carga autónomas

Juan David López Fuentes

Director(a): Diego Luis Linares Ospina

17 de mayo de 2024

Santiago de Cali, 17 de mayo de 2024.

Señores
Pontificia Universidad Javeriana Cali.
Ph.D. Luisa Rincón
Directora Maestría en Ingeniería de Software.
Cali.

Cordial Saludo.

Por medio de la presente hago constar que en mi calidad de director de trabajo de grado he revisado el proyecto titulado "Aplicar el agente inteligente basado en aprendizaje por refuerzo RELOAD para realizar pruebas de carga autónomas" realizado por el estudiante de Magister en Ingeniería de Software Juan David López Fuentes (cod: 8955312), el cual se encuentra terminado y considero que cumple con los requisitos para ser sustentado.
Atentamente,

Diego Luis Linares Ospina

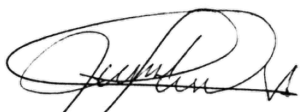
Santiago de Cali, 17 de mayo de 2024.

Señores
Pontificia Universidad Javeriana Cali.
Ph.D. Luisa Rincón
Directora Maestría en Ingeniería de Software.
Cali.

Cordial Saludo.

Me permito presentar a su consideración el proyecto de grado titulado "Aplicar el agente inteligente basado en aprendizaje por refuerzo RELOAD para realizar pruebas de carga autónomas" con el fin de cumplir con los requisitos exigidos por la Universidad y para que sea sometido a revisión del jurado y cumpla su aprobación, para conseguir posteriormente el título de Magister en Ingeniería de Software.

Atentamente,



Juan David López Fuentes
Código: 8955312

Ficha Resumen
Trabajo de Grado Maestría en Ingeniería de Software

TíTULO: Aplicar el agente inteligente basado en aprendizaje por refuerzo RELOAD para realizar pruebas de carga autónomas

1. Énfasis: Ingeniería de Software
2. Área de trabajo: Pruebas de carga y aprendizaje por refuerzo
3. Tipo de proyecto: Aplicado
4. Estudiante: Juan David López Fuentes
5. Correo electrónico: juanlopez92@javerianacali.edu.co
6. Dirección y teléfono: Villa mercedes casa 29, Popayán – Cauca, 3182543552
7. Director: Diego Luis Linares Ospina
8. Vinculación del director: Planta
9. Correo electrónico del director: dlinares@javerianacali.edu.co
10. Palabras clave: pruebas de carga, aprendizaje por refuerzo, prototipo, cargas autónomas, calidad de software
11. Fecha de inicio: 01/02/2024
12. Resumen: El presente proyecto tiene como objetivo aplicar el prototipo de agente inteligente basado en aprendizaje por refuerzo, llamado RELOAD que permita realizar pruebas de carga autónomas, para lo que se realizará una investigación con metodología cuantitativa de tipo experimental, donde se hará uso de un agente de prueba de carga impulsado por el aprendizaje por refuerzo propuesto que identifica los efectos de diferentes transacciones involucradas en la carga de trabajo y aprende cómo ajustar las transacciones para cumplir con el objetivo de la prueba. De esta manera, los resultados esperados implican utilizar un agente de pruebas de carga autónomo impulsado por el aprendizaje por refuerzo.

Agradecimientos

Quisiera comenzar expresando mi agradecimiento a todas las personas que han sido fundamentales en el desarrollo de este proyecto, ya que su colaboración, apoyo y aliento ha sido invaluable a lo largo de este proceso. En primer lugar, agradezco a mi familia y allegados por su incondicional apoyo emocional y por creer en mí en cada paso del camino, además quiero extender mi gratitud al director de este proyecto, Diego Luis Linares, por su orientación y capacidad para inspirar e impulsar el desarrollo de este proyecto, su profundo conocimiento ha sido una guía invaluable.

Asimismo, agradezco a la directora de maestría, Luisa Rincón y la profesora Susana Medina Gorillo por su valioso asesoramiento a lo largo de este proceso, sus sugerencias han enriquecido enormemente este trabajo y han contribuido a su calidad. Por último, pero no menos importante, agradezco a todos los profesionales y expertos en la materia que han contribuido con sus conocimientos y experiencias, su influencia ha sido esencial para el desarrollo de este proyecto.

En conclusión, este proyecto no habría sido posible sin el apoyo y la contribución de todas estas personas mencionadas anteriormente, por eso estaré siempre agradecido.

Resumen

El presente proyecto tiene como objetivo aplicar el prototipo de agente inteligente basado en aprendizaje por refuerzo, llamado RELOAD que permita realizar pruebas de carga autónomas, para lo que se realizará una investigación con metodología cuantitativa de tipo experimental, donde se hará uso de un agente de prueba de carga impulsado por el aprendizaje por refuerzo propuesto que identifica los efectos de diferentes transacciones involucradas en la carga de trabajo y aprende cómo ajustar las transacciones para cumplir con el objetivo de la prueba. De esta manera, los resultados esperados implican utilizar un agente de pruebas de carga autónomo impulsado por el aprendizaje por refuerzo.

Palabras Clave: pruebas de carga, aprendizaje por refuerzo, prototipo, cargas autónomas, calidad de software

Abstract

The objective of this project is to apply the prototype of an intelligent agent based on reinforcement learning, called RELOAD that allows autonomous load testing, for which research will be carried out with experimental quantitative methodology, where a powered load test agent will be used. by the proposed reinforcement learning that identifies the effects of different transactions involved in the workload and learns how to adjust the transactions to meet the test objective. In this way, the expected results involve using an autonomous load testing agent powered by reinforcement learning.

Keywords: load testing, reinforcement learning, prototype, autonomous charging, software quality

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Definición del problema

1.1.1. Planteamiento del problema

Muchos sistemas, desde sitios web de comercio electrónico hasta infraestructuras de telecomunicaciones, deben admitir el acceso simultáneo de cientos o miles de usuarios. Muchos de los problemas de campo están relacionados según Jiang et al. (2009) con sistemas que no se adaptan a las cargas de trabajo de campo en lugar de errores de funciones. De modo que según Sánchez Peño (2015) las pruebas de carga consisten principalmente en la medición del comportamiento del sistema con el propósito de aumentar la carga del mismo, ya sea por el número de peticiones que se efectúan en la WEB al mismo tiempo o por el número de usuarios que laboran de manera simultánea en un sistema, entre otros aspectos.

La eficiencia, como un elemento crucial de calidad, desempeña un papel fundamental en el éxito de los productos de software. Garantizar un rendimiento óptimo es especialmente vital en áreas donde asegurar tanto la calidad funcional como la no funcional del sistema es primordial. Se realiza un análisis exhaustivo del rendimiento con el fin de alcanzar objetivos clave, como medir las métricas de rendimiento, identificar problemas funcionales que puedan surgir bajo condiciones específicas, como una alta carga de trabajo, y detectar posibles incumplimientos de los requisitos de rendimiento. El modelado y las pruebas de rendimiento se consideran enfoques convencionales para alcanzar dichos objetivos en distintas fases del análisis de rendimiento.

El objetivo de una prueba de carga es según Flórez Tunaroza (2016) descubrir problemas funcionales y de rendimiento bajo carga. Los errores funcionales son típicamente fallos que no se manifiestan durante las pruebas funcionales. Ejemplos de estos problemas incluyen bloqueos del sistema y fallos en la gestión de la memoria cuando el sistema está sometido a una carga significativa. Los problemas de rendimiento a menudo se refieren a problemas de rendimiento como un alto tiempo de respuesta o un bajo rendimiento de carga.

En esta medida, Moghadam et al. (2021) asegura que existen diferentes enfoques comunes de prueba de carga, como técnicas que emplean código de fuente, análisis de modelos de sistemas y análisis de comportamiento. Sin embargo, la mayoría de técnicas impulsadas se basan principalmente en artefactos que no siempre están disponibles durante las pruebas. Por tanto, resulta crucial y aún constituye un desafío identificar los impactos en el rendimiento de las transacciones involucradas en la generación eficiente de una carga de trabajo efectiva.

Elaborar pruebas capaces de detectar fallos de rendimiento en sistemas de software extensos y complejos dentro de un plazo razonable representa un desafío considerable. Por un lado, se enfrenta a una amplia gama de combinaciones de valores de datos de entrada que deben ser exploradas. Por otro lado, la disponibilidad limitada de recursos para pruebas, así como la restricción de acceso al código fuente y a los detalles internos de estos sistemas, complican aún más esta tarea.

Chen and Hossain (2022) señalan que garantizar la calidad es esencial para un sistema de software exitoso. Los sistemas de software necesitan ser sometidos a pruebas en todas las fases del ciclo de vida de desarrollo de software (SDLC), sin importar el tipo de software en desarrollo. Si un error de software no se detecta en las etapas iniciales del SDLC, corregirlo en etapas posteriores resulta más complicado y costoso. De la misma manera, Flórez Tunaroza (2016) asegura que un prototipo adecuado para pruebas de carga minimiza el riesgo de tiempo de inactividad del sistema, puesto que las pruebas de carga permiten a los evaluadores simular el uso de aplicaciones en tiempo real bajo diferentes cargas.

Por otra parte, Navaei and Tabrizi (2022) aseguran que el aprendizaje por refuerzo en el contexto del desarrollo de software implica utilizar algoritmos para enseñar a los sistemas a tomar decisiones óptimas basadas en la retroalimentación recibida del entorno, lo que puede mejorar la calidad del software, además señalan que la aplicación del aprendizaje por refuerzo en el ciclo de vida de desarrollo de software (SDLC) permite a los sistemas automatizar tareas complejas, optimizar procesos y adaptarse dinámicamente a los cambios del entorno, mejorando así la eficiencia y la efectividad del desarrollo.

De esta manera, se destaca la relevancia del aprendizaje por refuerzo como un mecanismo de entrenamiento basado en retroalimentación para modelos de aprendizaje automático (ML). El cual se trata según Fan et al. (2023) de un agente de IA que participa en un entorno desconocido para lograr algunos objetivos predeterminados sin intervención humana, lo que le permite aprender las reglas del entorno complejo. Principalmente, mediante prueba y error en función de qué acción es recompensada o penalizada. En pocas palabras, la importancia de apoyarse en el aprendizaje por refuerzo radica en que el mismo obliga a un agente de IA

a poder descubrir de manera óptima las posibles mejores decisiones, definiendo el comportamiento correcto dentro de un entorno.

De acuerdo con Chen and Hossain (2022) la aplicación del aprendizaje automático en pruebas y control de calidad del software puede ayudar a los evaluadores en el proceso de prueba, incluida la detección temprana y la predicción de un error de software. El uso de técnicas de aprendizaje automático presenta nuevos desafíos para el proceso de pruebas y el aseguramiento de calidad. El aprendizaje automático, una rama de la inteligencia artificial, se enfoca en analizar conjuntos de datos para identificar patrones y tendencias. Se ha observado que ciertas actividades de prueba de software pueden ser concebidas como problemas de aprendizaje. Por lo tanto, el aprendizaje automático puede ser una herramienta eficaz para automatizar las pruebas de software, especialmente en sistemas altamente complejos.

Aplicar un agente inteligente basadado en aprendizaje por refuerzo que realice pruebas de carga autónomas fue el objetivo de este proyecto, permitiendo el uso de un algoritmo basado en Q-learning que permite generar cargas de trabajo eficientes para evaluar el cumplimiento de los requerimientos de rendimiento de un sistema en el desarrollo de software.

1.1.2. Formulación

El proyecto presentado tiene como finalidad abordar principalmente a la siguiente pregunta:

- ¿Cómo Aplicar un prototipo de agente inteligente basado en aprendizaje adaptativo que permita realizar pruebas de carga autónomas?

1.1.3. Sistematización

- ¿Cómo definir el ambiente para implementar un algoritmo Q-learning basado en el aprendizaje por refuerzo?
- ¿Cómo se establecen los parámetros (acciones, estados y recompensas) de un agente por refuerzo?
- ¿Cómo implementar una prueba de carga mediante un agente que utiliza la técnica de aprendizaje por refuerzo?
- ¿Cómo evaluar el rendimiento del agente que realiza una prueba de carga con aprendizaje por refuerzo y sus resultados respecto a las pruebas de carga tradicionales?

1.2. Objetivos del proyecto

1.2.1. Objetivo General

Aplicar un agente inteligente basado en aprendizaje por refuerzo que permita realizar pruebas de carga autónomas.

1.2.2. Objetivos específicos

1. Definir el ambiente para implementar un algoritmo Q-Learning basado en el aprendizaje por refuerzo.
2. Establecer los parámetros (acciones, estados y recompensas) de un agente por refuerzo.
3. Implementar una prueba de carga mediante un agente que utiliza la técnica de aprendizaje por refuerzo.
4. Evaluar el rendimiento del agente que realiza una prueba de carga con aprendizaje por refuerzo y sus resultados respecto a las pruebas de carga tradicionales.

1.3. Delimitaciones y alcances

El alcance de este proyecto fue emplear un agente inteligente de pruebas de carga impulsado por aprendizaje por refuerzo, RELOAD, para realizar pruebas de carga a un sistema sujeto a pruebas (SUT) o software bajo prueba. En estas pruebas, la carga de trabajo se configura como un conjunto de usuarios (virtuales) que llevan a cabo diversas transacciones en el software que está siendo evaluado y de esta manera obtener resultados sobre los tiempos de respuesta y la tasa de error, que son criterios importantes para evaluar el rendimiento del sistema.

1.4. Justificación

Las pruebas de carga, en general, se refieren a la práctica de evaluar el comportamiento de un sistema bajo carga. La prueba de carga evalúa si un sistema de software en producción puede manejar efectivamente las condiciones de carga real sin afectar su rendimiento previsto. Esta forma, las pruebas de carga, ayudan a detectar posibles cuellos de botella en las aplicaciones al simular usuarios y las transacciones simultáneas bajo diversas condiciones de carga. Estas pruebas se pueden realizar en un entorno que simule el entorno de producción o directamente en el entorno de producción con configuraciones específicas. La carga puede definirse de varias maneras, como la cantidad de solicitudes y usuarios simultáneos. En última instancia, las pruebas de carga ayudan a mejorar el rendimiento del sistema antes de implementar el código en entornos de producción.

Según Gil-Vera and Seguro-Gallego (2022) las pruebas de carga son fundamentales para garantizar el rendimiento y la confiabilidad de los sistemas de software de producción, así como ayuda a identificar la debilidad que afecta el rendimiento esperado del sistema. De esta manera, uno de los principales problemas ante la no realización de pruebas de carga de una base de datos, puede dar lugar a problemas de rendimiento que no se detectan hasta que la aplicación está en funcionamiento, lo que conduce a la frustración del usuario, a la pérdida de datos y la posible pérdida de ingresos. Todos estos problemas se pueden evitar simplemente probando la base de datos antes del lanzamiento del sistema.

Al respecto, Gil-Vera and Seguro-Gallego (2022) afirman que las pruebas de carga se pueden utilizar de varias maneras diferentes para identificar problemas de rendimiento de la base de datos. Las pruebas de carga proporcionan datos importantes sobre cómo funcionan las aplicaciones al examinar medidas como el tiempo de respuesta, la frecuencia de errores, el rendimiento, la utilización de recursos como la CPU y la memoria, la cantidad de solicitudes por segundo y los tiempos de espera en la red. Las pruebas ofrecen múltiples beneficios,

incluyendo la identificación y solución de problemas de rendimiento, el fortalecimiento de la capacidad de ajuste, el incremento en la satisfacción del usuario, y la disminución de gastos innecesarios.

En el campo de las Aplicaciones Web, el Q-learning (algoritmo de aprendizaje por refuerzo) puede ser un elemento clave para generar sistemas de prueba de carga. Debido a los efectos que se generan en las mismas a causa de la importante carga de usuarios que manejan dichas aplicaciones. Por lo cuál a partir de las pruebas de rendimiento y de software automatizadas es posible determinar el límite superior de todos los componentes principales de las mismas, como la base de datos, las restricciones de la red, la capacidad de los servidores, etc.

De esta manera, las pruebas de carga automatizadas podrían ser un aliado importante para generar beneficios de diversa índole, tanto en la representatividad de los costos organizacionales, como en la satisfacción de clientes y en la minimización de riesgos, entre otros beneficios. En esta medida, el presente trabajo representa un importante potencial para reducir costos y esfuerzo humano, puesto que aún existen diferentes aspectos de las pruebas de software que son difíciles de automatizar. El presente trabajo permitió abordar la generación automatizada de casos de prueba de rendimiento que dependen del uso de código fuente o análisis de modelos de sistemas o técnicas fundamentados en casos de uso.

Algunas de las ventajas del desarrollo del presente trabajo es el uso de la automatización inteligente, donde es posible la generación eficiente de casos de prueba y reducción del tiempo de cálculo, así como ya se mencionó, la menor dependencia del código fuente y los modelos. Su aplicabilidad puede ser beneficioso para las pruebas de variantes de software y las pruebas de rendimiento de regresión, lo que a su vez podría generar un enfoque para admitir casos de prueba de rendimiento basados en cargas de trabajo.

Singh (2022) asegura que por el lado de la minimización de costos se considera que identificar problemas de rendimiento en una etapa temprana es significativamente menor en comparación con etapas posteriores. Por ende, se considera que las pruebas de carga eficientes y oportunas permiten detectar inconvenientes y problemas previamente a la generación de costos por eliminación de errores. De la misma manera, a partir del desarrollo del proyecto es posible conducir a una reducción de costos para la generación de casos de prueba de rendimiento al reutilizar la política aprendida en el Software Bajo Prueba (SUT).

Por otro lado, Whiting and Datta (2021) considera que las pruebas de carga automatizadas son un punto clave para generar escalabilidad mejorada, lo que puede resolver incon-

venientes vinculados al manejo del software, específicamente, cuando el mismo no maneja adecuadamente a los usuarios simultáneos, lo que genera problemas como una mala utilización de los recursos y pérdidas de memoria. De este modo, las pruebas de carga permiten a los equipos de software descubrir los límites de la capacidad operativa de su aplicación, ya que cuando el equipo es consciente de las limitaciones del sistema, puede identificar fácilmente los cambios necesarios para hacer que el producto sea más escalable. Cabe mencionar que, a raíz de su funcionamiento, pueden contribuir de manera positiva a las empresas para desarrollar acuerdos de nivel de servicio para sus productos. En resumen, las pruebas de carga permiten identificar las deficiencias de los productos, de modo que estos puedan ser ajustados para mejorar su rendimiento y características determinadas.

Finalmente, es importante aclarar que el desarrollo de este proyecto es de gran relevancia, puesto que el aprendizaje por refuerzo sin modelos se utiliza ampliamente para encontrar el comportamiento óptimo para lograr un objetivo en muchos problemas de toma de decisiones sin depender de un modelo del sistema. En esta medida el agente de prueba puede ser una manera óptima de generar la carga de trabajo cumpliendo con la automatización de pruebas eficientes que serían posibles sin depender de modelos de sistema o código fuente, con lo que además se pretende lograr el objetivo de la prueba con un menor esfuerzo en comparación con los procedimientos de prueba de carga comunes y generar mayor eficiencia en el proceso.

1.5. Metodología

El presente proyecto se basó en una investigación cuantitativa de tipo experimental debido a que se basa en la aplicación de un agente impulsado por aprendizaje por refuerzo aplicado a pruebas de carga. En resumen, implicó emplear el agente RELOAD propuesto, el cual detecta los impactos de varias transacciones en la carga de trabajo y aprende a modificarlas para cumplir con el objetivo de la prueba.

En primer lugar, se realizó un análisis de manera general sobre Q-learning, aprendizaje por refuerzo y RELOAD para poder establecer los parámetros del agente y determinar el ambiente necesario, una vez definidos se realizaron pruebas de carga a través de diferentes experimentos para evaluar la eficiencia del agente sobre los tiempos de respuestas y transacciones concurrentes, para por último realizar la evaluación de los resultados obtenidos en el proceso de pruebas de carga.

Con base en lo anterior, se abordaron las siguientes actividades para cumplir con los objetivos del proyecto:

1.5.1. Establecer los parámetros (acciones, estados y recompensas) de un agente por refuerzo

- Explorar la técnica de aprendizaje por refuerzo y el algoritmo Q-learning
- Revisar las características del agente de pruebas de carga.
- Identificar los parámetros necesarios que el agente utiliza, como acciones, estados y recompensas.
- Identificar los elementos que el agente tiene en cuenta para realizar acciones y aprender la política óptima.
- Revisar la arquitectura del agente y como funciona.

1.5.2. Definir el ambiente para implementar un algoritmo Q-Learning basado en el aprendizaje por refuerzo

- Analizar el agente que se pretende entrenar y con ello establecer el entorno donde interactúa el agente.
- Definir el sistema sujeto a pruebas que va ser el entorno para la prueba.
- Identificar las funcionalidades y transacciones del sistema sujeto a pruebas
- Definir donde se va alojar el ambiente y el agente de pruebas.
- Crear el plan de pruebas en la herramienta JMeter para las transacciones del sistema sujeto a pruebas.
- Configurar el agente para soportar el sistema sujeto a pruebas y sus transacciones

1.5.3. Implementar una prueba de carga mediante un agente que utiliza la técnica de aprendizaje por refuerzo

- Diseñar una serie de experimentos para evaluar la eficiencia del agente de pruebas.
- Realizar pruebas de manera empírica a través de la ejecución de experimentos al sistema sujeto a pruebas establecido.
- Ejecutar el agente con un objetivo distinto al aprendido para efectuar la fase de aprendizaje por transferencia.

1.5.4. Evaluar el rendimiento del agente que realiza una prueba de carga con aprendizaje por refuerzo y sus resultados respecto a las pruebas de carga tradicionales

- Realizar una evaluación de la ejecución de los experimentos realizados al sistema sujeto a pruebas establecido con los diferentes parámetros definidos.
- Evaluar el resultado de la ejecución de pruebas con el agente, respecto a una prueba de carga tradicional.
- Analizar los resultados obtenidos de las diferentes ejecuciones de prueba.

1.6. Resultados obtenidos

Los resultados obtenidos en el proyecto fueron altamente satisfactorios y prometedores. A través del uso del agente inteligente Reload para llevar a cabo pruebas de carga, se logró una mejora significativa en la eficiencia y la efectividad del proceso de pruebas de rendimiento. Durante el análisis y la configuración del agente, se identificaron algunos desafíos iniciales debido a la falta de documentación y la complejidad inherente del código fuente, lo que podría haber obstaculizado la comprensión y la implementación efectiva del mismo. Sin embargo, al superar estos obstáculos y lograr una configuración exitosa del agente, se observaron numerosos beneficios.

En primer lugar, el uso de Reload permitió una generación eficiente de cargas de trabajo de prueba adaptadas a las necesidades específicas del sistema bajo prueba, reduciendo así la dependencia de modelos complejos y del código fuente. Además, la capacidad de adaptación del agente a cambios en el entorno de ejecución y su habilidad para reutilizar políticas aprendidas en diferentes escenarios de prueba proporcionaron una mayor flexibilidad y eficiencia en el proceso de pruebas.

En términos de resultados concretos, se observó una mejora significativa en la eficacia de las pruebas de carga realizadas con Reload en comparación con las pruebas de carga tradicionales. Estas mejoras se reflejaron en una detección más precisa de problemas de rendimiento, una mayor capacidad para optimizar la carga de trabajo de prueba y una reducción en el esfuerzo asociado con las pruebas de rendimiento.

En resumen, los resultados del proyecto demostraron claramente que el uso del agente inteligente Reload impulsado por aprendizaje por refuerzo ofrece una serie de ventajas significativas en el proceso de pruebas de rendimiento. Estos resultados respaldan la eficacia

y el potencial del enfoque adoptado, consolidando así la importancia del aprendizaje por refuerzo en el campo de la ingeniería de software y destacando la relevancia de Reload como una herramienta valiosa en el ciclo de vida de desarrollo de software.

Capítulo 2

Marco de referencia

En este capítulo se presentan las bases teóricas y los trabajos previos que sirven como base para la realización del proyecto.

2.1. Marco Teórico

2.1.1. Pruebas de rendimiento

Las pruebas de rendimiento, según Whiting and Datta (2021), son una forma de evaluación que mide la velocidad, capacidad de respuesta y estabilidad de una computadora, red, software o dispositivo bajo una carga de trabajo específica. Las organizaciones realizan estas pruebas para detectar posibles cuellos de botella relacionados con el rendimiento.

Según Abdul et al. (2020), el propósito de las pruebas de rendimiento radica en detectar y eliminar los puntos críticos de rendimiento en las aplicaciones de software, contribuyendo así a asegurar la calidad del software. Sin pruebas de rendimiento, el sistema podría experimentar tiempos de respuesta lentos y una experiencia poco consistente para los usuarios y el sistema operativo.

Como resultado, se genera una experiencia negativa para el usuario en general. Las pruebas de rendimiento son útiles para verificar si un sistema desarrollado cumple con los estándares de velocidad, capacidad de respuesta y estabilidad bajo diferentes cargas de trabajo, contribuyendo así a mejorar la experiencia del usuario. Es importante resaltar que estas pruebas deben llevarse a cabo después de completar las pruebas funcionales.

Por otro lado, Lawanna (2012) plantea elementos asociados al diseño de las fases de pruebas de software, las cuales se pueden dividir en tres fases principales: pruebas preliminares, pruebas y pruebas de aceptación del usuario. En cuanto a la fase de prueba preliminar se afirma que la misma puede llevarse a cabo especialmente para que los probadores aclaren los requisitos de especificación del cliente. De acuerdo con esto, las pruebas de software pueden satisfacer las necesidades tanto de los evaluadores como de los clientes. En este sentido, la fase de prueba preliminar se presenta acorde con los siguientes elementos:

- **Revisar la especificación de requisitos**: Este paso es importante porque muchos clientes inicialmente brindan información insuficiente como el tamaño y costo del proyecto. Entonces, el documento esencial requerido se conoce como especificación de requisitos, que consiste en una explicación de las funciones que el sistema debe cumplir.

- **Elaboración del plan de pruebas**: En este paso se crea un documento que detalla los recursos, el calendario de actividades de prueba, el alcance y la estrategia de prueba, incluyendo elementos como características, tareas, entorno, habilidades del evaluador, criterios de diseño y riesgos.

- **Configuración de la herramienta de software**: Este paso se lleva a cabo una vez que se tienen claros los diferentes módulos del producto, así como el hardware y el sistema operativo requeridos.

- **Preparación del entorno de pruebas**: Aquí se identifican los componentes del entorno de producción que deben someterse a pruebas, y se establece un entorno de pruebas adecuado para estos elementos, que pueden incluir aplicaciones comerciales, herramientas administrativas, hardware, bases de datos, servicios, sistemas de seguridad, aplicaciones y sistemas de red, entre otros.

- **Creación de casos de prueba**: En este paso se elabora un documento que describe la entrada y la respuesta esperada para una característica específica de la aplicación.

- **Configuración de herramientas de automatización de pruebas**: Este paso implica la planificación de una estrategia de prueba para automatizar pruebas de software, como, por ejemplo, la ejecución de casos de prueba para pruebas de regresión.

- **Selección de la herramienta de prueba de aceptación**: En este paso se decide qué herramienta de prueba de aceptación se utilizará para garantizar que las pruebas de software cumplan con los requisitos de la especificación.

Por otro lado, la fase de prueba puede ser una fase separada que la lleva a cabo un equipo de prueba diferente una vez completada la implementación. La técnica de prueba se selecciona en función de la perspectiva del equipo de prueba. En este sentido la fase de prueba se divide en tres pasos:

- **Verificación**: Las actividades de verificación comprenden revisiones técnicas, inspecciones de software, recorridos y acciones destinadas a confirmar los requisitos del software (por ejemplo, asegurarse de que estén alineados con las especificaciones), verificar los componentes de diseño (asegurarse de que estén en línea con el software), validar los

requisitos de comportamiento, llevar a cabo pruebas unitarias, pruebas de integración, pruebas del sistema, verificar las pruebas de aceptación y realizar auditorías. Básicamente, implica determinar la corrección de la implementación del proceso correcto, como evaluar si el software está construido adecuadamente.

- **Validación**: Se enfoca en comprobar si el software desarrollado cumple con los requisitos del usuario.

- **Pruebas**: Este enfoque resulta beneficioso tanto para confirmar como para validar. Además, otras estrategias eficaces para la confirmación abarcan el análisis estático, las revisiones, las inspecciones y las sesiones de revisión.

Las tres fases están diseñadas para describir todas las actividades relacionadas con las pruebas de software. De acuerdo con esto, los probadores de software pueden aprovechar el diseño para implementar las ideas de las pruebas de software y abordar los problemas críticos.

2.1.2. Pruebas de Carga

De acuerdo con Irrazábal and Mascheroni (2022) la prueba de carga promedio evalúa cómo funciona su sistema en condiciones normales esperadas. Las pruebas de estrés evalúan cómo se desempeña un sistema en sus límites cuando la carga excede el promedio esperado. Las pruebas de inmersión evalúan la confiabilidad y el rendimiento de su sistema durante períodos prolongados.

La prueba de carga, según Peñafiel (2022), es una evaluación no funcional del software donde se examina cómo funciona la aplicación bajo una carga determinada. Se busca entender su desempeño cuando múltiples usuarios la utilizan al mismo tiempo. El propósito principal de estas pruebas es identificar y resolver posibles problemas de rendimiento, asegurando así que la aplicación funcione de manera estable y eficiente antes de ser lanzada al público.

De acuerdo con la fuente citada, las pruebas de carga suelen identificar aspectos como: la cantidad máxima de operaciones que puede manejar una aplicación, verificar si la infraestructura actual puede soportar la aplicación en funcionamiento; la capacidad de la aplicación para mantenerse estable con una carga máxima de usuarios, es decir, el número de usuarios simultáneos que puede atender, y su capacidad de adaptarse para permitir el acceso de más usuarios. En el ámbito de la Ingeniería de Software, estas pruebas son comúnmente utilizadas para evaluar aplicaciones Cliente/Servidor basadas en la Web, ya sea dentro de una red

interna o en Internet.

De acuerdo con Jiang (2015) algunas de las estrategias de prueba de carga se resumen a las siguientes:

- **Pruebas de carga manuales**: Esta técnica consiste en llevar a cabo pruebas de carga de forma manual, pero presenta limitaciones significativas. No garantiza resultados consistentes, no puede aplicar niveles de estrés medibles a una aplicación y resulta difícil de coordinar.

- **Herramientas de prueba de carga desarrolladas internamente**: Las organizaciones que reconocen la importancia de las pruebas de carga pueden crear sus propias herramientas para llevar a cabo estas pruebas según sus necesidades y requerimientos específicos.

- **Herramientas de prueba de carga de código abierto**: Existen diversas herramientas de prueba de carga disponibles como software de código abierto, las cuales son gratuitas. Aunque pueden no ser tan avanzadas como las opciones de pago, son una opción viable, especialmente para aquellos con recursos financieros limitados.

- **Herramientas de prueba de carga de nivel empresarial**: Estas herramientas suelen contar con características de grabación y reproducción. Tienen la capacidad de trabajar con una amplia variedad de protocolos y pueden simular un gran número de usuarios, lo que las convierte en una opción valiosa para empresas y proyectos de gran escala.

Por otro lado, Ortiz and Duque (2022) afirman que existen tipos básicos de pruebas de carga: prueba de capacidad, prueba de estrés, pruebas de inmersión, pruebas de picos y prueba de volumen:

- **Pruebas de capacidad**: Es una prueba diseñada para mostrar si el sitio web o la aplicación pueden soportar la cantidad de estrés para la que han sido programados y dónde puede tener problemas el sitio. Puede resultar muy beneficiosa al intentar identificar cuellos de botella o problemas con el código. Al realizar una prueba de capacidad, hay algunas cosas a considerar. En primer lugar, se deben establecer criterios adecuados para garantizar que los resultados sean precisos y reflejen escenarios de la vida real. Esto se puede hacer revisando los niveles de tráfico existentes y estimando posibles aumentos futuros o aumentos repentinos. También puede automatizar las pruebas de carga para ejecutarlas en diferentes momentos o al mismo tiempo.

- **Pruebas de estrés**: Permiten probar el límite máximo al que llegará un sitio web antes de que falle y se cierre, también permite comprender cómo funcionará un sitio web bajo una carga intensa y posibilita planificar tanto los picos esperados como las circunstancias imprevistas, como una publicación de blog que de repente se vuelve viral. Las pruebas de estrés se pueden utilizar para probar el nivel de seguridad de un sistema, protegiéndolo contra ataques dañinos. Hay algunos pasos a seguir para realizar una prueba de estrés.

- **Pruebas de inmersión**: Se realiza una prueba de inmersión para evaluar el rendimiento de un sitio web durante un período de tiempo prolongado. Los usuarios se presentan gradualmente y se puede ver cómo se desempeña un sitio web con carga adicional a lo largo del tiempo. Estas pruebas permiten a los desarrolladores identificar qué tipos de pérdidas de memoria, degradación y otras fallas del sistema ocurren con el tiempo.

- **Pruebas de picos**: En este tipo de pruebas, la herramienta de prueba de carga genera un aumento repentino en el número de usuarios para ver cómo se comporta el sitio web. Las pruebas de picos no solo prueban un aumento en el número de usuarios, sino que también generan resultados basados en una disminución en el número.

- **Pruebas de volumen**: Las pruebas de volumen se encargan de agregar un gran volumen de datos. Con las pruebas de volumen es posible estudiar el tiempo de respuesta.

2.1.3. Inteligencia Artificial

De acuerdo con Ponce et al. (2014), la Inteligencia Artificial (IA) nació, puesto que las computadoras pudieron almacenar más información y se volvieron más rápidas, más baratas y más accesibles. Los avances en los algoritmos de aprendizaje automático también se perfeccionaron, lo que permitió a las personas elegir con mayor precisión qué algoritmo utilizar para resolver sus problemas. Demostraciones tempranas, como el General Problem Solver de Newell y Simon y ELIZA de Joseph Weizenbaum, mostraron un potencial prometedor en la resolución de problemas y en la interpretación del lenguaje hablado, respectivamente.

De acuerdo con Mikalef et al. (2019) estos éxitos, así como la defensa de los principales investigadores (es decir, los asistentes al DSRPAI), convencieron a agencias gubernamentales como la Agencia de Proyectos de Investigación Avanzada de Defensa (DARPA) para financiar investigación de Inteligencia Artificial en varias instituciones. El gobierno mostraba un interés particular en una máquina capaz de transcribir y traducir el lenguaje hablado, así como en el procesamiento de datos de alto rendimiento. Las expectativas eran altas y el optimismo reinaba. En 1970, Marvin Minsky declaró a la revista Life que .[en] un plazo de tres a ocho años, tendríamos una máquina con la inteligencia general de un humano promedio".

Sin embargo, aunque se había establecido un fundamento básico, aún quedaba un largo camino por recorrer antes de alcanzar los objetivos finales en áreas como el procesamiento del lenguaje natural, el pensamiento abstracto y la autoconciencia.

En los años 80, la inteligencia artificial experimentó un renacimiento impulsado por dos factores principales: la ampliación del conjunto de herramientas algorítmicas y un aumento en la financiación. John Hopfield y David Rumelhart popularizaron las técnicas de .ªprendizaje profundo", que permitían a las computadoras aprender de la experiencia. Por otro lado, Edward Feigenbaum introdujo los sistemas expertos, que emulaban el proceso de toma de decisiones de un experto humano. Estos sistemas consistían en programas que consultaban a expertos en un campo específico sobre cómo responder ante diversas situaciones, y una vez que se había capturado ese conocimiento para la mayoría de las circunstancias, personas sin experiencia podían recibir orientación de estos programas. Los sistemas expertos se adoptaron ampliamente en diversas industrias.

El gobierno japonés proporcionó un fuerte respaldo financiero a los sistemas expertos y otros proyectos relacionados con la inteligencia artificial como parte de su Proyecto de Computación de Quinta Generación (FGCP). Entre 1982 y 1990, se invirtieron 400 millones de dólares con el propósito de transformar el procesamiento informático, implementar la programación lógica y avanzar en el campo de la inteligencia artificial. Desafortunadamente, la mayoría de los ambiciosos objetivos no se lograron. No obstante, se podría argumentar que los efectos indirectos del FGCP inspiraron a una joven generación talentosa de ingenieros y científicos. Sin embargo, la financiación del FGCP se detuvo y la inteligencia artificial perdió su posición de prominencia.

Por otro lado, según Rouhiainen (2018), muchas de las metas históricas de la inteligencia artificial fueron alcanzadas durante las décadas de 1990 y 2000. En 1997, Deep Blue de IBM, un programa de computadora diseñado para jugar ajedrez, derrotó al entonces campeón mundial de ajedrez, Gary Kasparov. Este evento, ampliamente publicitado, marcó la primera vez en la historia en la que un campeón mundial de ajedrez perdió ante una máquina, representando así un importante avance hacia un programa de toma de decisiones artificialmente inteligente. En ese mismo año, Dragon Systems implementó software de reconocimiento de voz en Windows, otro avance significativo en el campo de la interpretación del lenguaje hablado.

Al respecto, el autor plantea que en la actualidad la definición que se otorga a la Inteligencia Artificial es la habilidad que tienen los ordenadores para hacer actividades que normalmente requieren de la inteligencia humana, así como también se la asocia con la capa-

cidad de las máquinas para emplear algoritmos, aprender de los datos y utilizar lo aprendido en la toma de decisiones tal y como lo haría un individuo, no obstante, a diferencia de las personas, los dispositivos fundamentados en Inteligencia Artificial no requieren de descanso y tienen la característica de analizar grandes volúmenes de información simultáneamente.

Según Thongprasit and Wannapiroon (2022) la inteligencia artificial es proyectada a futuro como una herramienta de gran impacto, pues en la actualidad los científicos en inteligencia artificial han logrado avances trascendentales en todos los campos. Se afirma que la ayuda de la IA puede ser significativa para los seres humanos, incluso en la capacidad de ver, oír y entender perfectamente.

2.1.4. Aprendizaje por refuerzo

Según Lascorz and Alcalá (2018), el aprendizaje por refuerzo es uno de los temas más debatidos, seguidos y considerados en el campo de la inteligencia artificial (IA). Este método implica la toma de decisiones y consiste en aprender el comportamiento óptimo en un entorno con el fin de maximizar la recompensa. Es un enfoque de entrenamiento de aprendizaje automático que recompensa los comportamientos deseados y penaliza los no deseados. En términos generales, un agente de aprendizaje por refuerzo, es decir, la entidad que se está entrenando, es capaz de percibir y comprender su entorno, tomar acciones y aprender a través de la experiencia.

De acuerdo con Rudkowskyj Hernanz (2019), el aprendizaje por refuerzo se cuenta entre los métodos empleados por los desarrolladores para instruir sistemas de aprendizaje automático. Su relevancia radica en su capacidad para capacitar a un agente, sea una función en un videojuego o un robot en un entorno industrial, para adaptarse y desenvolverse en las complicaciones del entorno específico para el cual fue concebido.

En esta medida, Lascorz and Alcalá (2018) afirma que el aprendizaje por refuerzo se basa en la lógica de la teoría del aprendizaje por prueba y error, donde el programador no conoce completamente las acciones que debería efectuar el programa o agente, solo tiene la facultad de decidir cuándo se ha completado la tarea o no. En esta medida, su funcionamiento se sintetiza en varios elementos, entre lo que se destaca la interacción del agente con el entorno y la ejecución de una acción. Posteriormente, dicha acción afecta al estado del agente en el entorno, donde la recompensa se limita a una señal determinando la conveniencia o no de las acciones efectuadas y a partir de tal indicación se busca el mejoramiento del comportamiento posterior.

En resumen, Lascorz and Alcalá (2018) aclara que en el aprendizaje por refuerzo se busca encontrar un comportamiento óptimo que propenda por la resolución del problema aprendiendo de las acciones pasadas y buscando la maximización de la recompensa acumulada en el tiempo. No obstante, las recompensas no se muestran de manera momentánea y buscan la maximización acumulada, por lo cual el agente debe razonar y elegir las acciones con las que a largo plazo logre un mejor resultado.

Entorno al aprendizaje por refuerzo se ha establecido el dilema de explorar y explotar, donde Poole and Mackworth (2010) plantea que este se fundamenta en que, si un agente ha elaborado un buen curso de acciones, debería optar por dar continuidad a dichas acciones, de manera que pueda explotar lo que ha determinado o en cambio debería explorar más para encontrar mejores acciones. De esta manera, el dilema radica en que un agente que nunca realiza exploración puede actuar siempre de una manera que podría ser mejor a si hubiera explorado antes, en tanto que, un agente que siempre explora probablemente nunca usara lo que ha aprendido, así mismo se cree que la exploración puede generar daños irreversibles.

En conclusión, el aprendizaje por refuerzo (RL), representa una rama dentro del ámbito del aprendizaje automático, donde un agente se capacita para tomar decisiones a través de la interacción con su entorno con el fin de lograr un objetivo específico. En este proceso, el agente va adquiriendo conocimiento sobre el comportamiento óptimo a través de la experiencia, enfrentándose a un proceso de prueba y error donde recibe retroalimentación en forma de recompensas o penalizaciones según las acciones que emprende. El objetivo primordial del agente radica en maximizar la recompensa acumulada a lo largo de su interacción continua con el entorno. A continuación, se detallan los elementos fundamentales para el aprendizaje por refuerzo:

- **Agente:** El agente es la entidad que interactúa con el entorno y toma decisiones. Puede ser un robot, un programa de computadora o cualquier otro sistema capaz de percibir su entorno y actuar en consecuencia. El agente utiliza información del entorno para seleccionar acciones que maximicen una señal de recompensa a largo plazo.

- **Entorno:** El entorno es el contexto en el que el agente opera y toma decisiones. Puede ser físico, como un laberinto para un robot, o virtual, como un videojuego o un sistema o aplicación. El entorno determina las acciones disponibles para el agente, así como las recompensas o castigos que el agente recibe en respuesta a sus acciones.

- **Acciones:** Las acciones son las decisiones que el agente puede tomar en un determinado estado del entorno. Estas acciones pueden ser discretas, como moverse en una dirección

específica, o continuas, como ajustar la velocidad de un motor. El conjunto de acciones disponibles para el agente en cada estado define el espacio de acciones del problema.

- **Estados:** Los estados representan las situaciones o condiciones en las que se encuentra el agente en el entorno. Estos estados pueden ser estados físicos, como la posición y la velocidad de un robot, o estados abstractos, como la configuración actual de un juego de mesa. El agente utiliza información del estado actual para decidir qué acción tomar a continuación.

- **Recompensas:** Las recompensas son señales numéricas que el agente recibe del entorno en respuesta a sus acciones. Estas recompensas representan la bondad de las acciones tomadas por el agente y se utilizan para guiar el proceso de aprendizaje. El objetivo del agente es maximizar la cantidad total de recompensas acumuladas a lo largo del tiempo.

- **Política:** La política es la estrategia que el agente sigue para seleccionar acciones en función de los estados del entorno. Puede ser determinista, lo que significa que selecciona una acción específica en cada estado, o estocástica, lo que significa que selecciona acciones de acuerdo con una distribución de probabilidad. El objetivo del agente es aprender una política óptima que maximice la recompensa total a largo plazo.

Por otra parte, el proceso de toma de decisiones de Markov se utiliza como fundamento para los sistemas de aprendizaje mediante refuerzo. En este proceso, un agente se encuentra en una situación específica dentro de un entorno y debe elegir la acción más adecuada entre varias opciones disponibles en ese momento. Algunas de estas acciones ofrecen recompensas como incentivo. Según Tappler et al. (2019), los procesos de decisión de Markov permiten representar sistemas reactivos con respuestas que siguen ciertas probabilidades. Durante la ejecución, el entorno selecciona y ejecuta acciones de forma no determinista, ante las cuales el sistema responde en función de su estado actual y una función de transición de probabilidad. De este modo, el sistema modifica su situación y produce una salida.

De la misma manera, se presenta Q-learning como uno de los algoritmos mas prominentes en el aprendizaje por refuerzo. De acuerdo con Boussakssou et al. (2020) existen varios algoritmos disponibles para aprendizaje reforzado, siendo los algoritmos Q-Learning unos de los algoritmos RL más utilizados, con la principal ventaja de su simplicidad. Necesitan una cantidad mínima de operaciones matemáticas y, en su forma básica, pueden ser representados mediante ecuaciones sencillas y ejecutados sin dificultad en programas informáticos. Los contextos de aprendizaje por refuerzo, como el Q-learning, se caracterizan por estados, acciones y recompensas, que pueden ser positivas o negativas.

Q-Learning es un algoritmo de aprendizaje por refuerzo que permite que un agente aprenda a tomar decisiones óptimas en un entorno desconocido. Este algoritmo se basa en la idea de aprender una función de valor de acción óptima, denotada como *Q(s,a)* que representa la utilidad esperada de tomar una acción a en un estado s.

En Q-Learning, dos elementos clave son la tabla Q y la fórmula de actualización de la tabla Q. La tabla Q es una matriz que almacena los valores asociados a cada par de estado-acción, representando cuánto valor espera el agente de tomar una acción en un estado específico. La fórmula de actualización de la tabla Q determina cómo se ajustan estos valores en función de la retroalimentación recibida del entorno, permitiendo al agente aprender y mejorar su comportamiento a lo largo del tiempo mediante la comparación entre las recompensas obtenidas y las estimaciones previas de los valores de las acciones.

Q-Learning es según Li et al. (2020) una forma de aprendizaje por refuerzo sin modelo. El área de estudio comprende un agente con distintos estados y un conjunto de acciones disponibles para él. El agente puede cambiar de un estado a otro al tomar una acción específica. Cada vez que el agente hace esta transición, es decir, pasa a un nuevo estado, recibe una recompensa. El objetivo del agente es maximizar la suma total de estas recompensas. Para lograr esto, el agente debe tomar las mejores decisiones posibles en cada estado. Para ayudar en este proceso, se utiliza una función llamada función Q, que calcula la calidad de cada posible combinación estado-acción. Inicialmente, esta función proporciona un valor inicial. Sin embargo, a medida que el agente interactúa con su entorno y obtiene recompensas, estos valores se ajustan para reflejar las vivencias del agente. Este procedimiento facilita que el agente aprenda la mejor manera de actuar en diversas circunstancias a lo largo del tiempo. En resumen, La fórmula principal del Q-learning para actualizar el Q-valor se basa en la idea de actualizar el valor de *Q(s,a)* basado en la recompensa obtenida y el valor máximo esperado de los Q-valores del próximo estado:

$$Q(s,a) = Q(s,a) + \alpha[r + \gamma max_{a'} Q(s',a') - Q(s,a)] \tag{2.1}$$

A continuación, se detalla cada componente de la formula de actualización Q-valor:

- *Q(s,a)* es el valor de Q en el estado s y la acción a.
- α es la tasa de aprendizaje, que determina la rapidez con la que se actualizan los valores de la tabla Q.
- γ es el factor de descuento, que determina la importancia relativa de las recompensas futuras.

- r es la recompensa recibida por tomar la acción a en el estado s.
- s' es el próximo estado después de tomar la acción a.
- a' es la siguiente acción que el agente selecciona en el próximo estado s'.

Es importante mencionar que este algoritmo se identifica como un algoritmo fuera de política, dado que no se basa en una política predeterminada para tomar decisiones, sino que aprende directamente de la experiencia adquirida en el entorno. El algoritmo de entrenamiento de Q-learning implica iterar sobre los estados y acciones, actualizando los Q-valores en cada paso hasta que el agente haya aprendido suficiente, como se puede ver a continuación:

1. **Inicialización:** Inicializa la tabla Q con valores arbitrarios. Para cada estado s y acción a, establece $Q(s,a)$ en 0 o algún valor inicial aleatorio.
2. **Para cada episodio:**
   - Inicializa el estado actual s del entorno.
   - Mientras no se haya alcanzado un estado final:
     - Selecciona una acción a utilizando una política de selección de acción, como ε-greedy, que equilibra entre la exploración y la explotación.
     - Ejecuta la acción a y observa el próximo estado s' y la recompensa r.
     - Actualiza la tabla Q utilizando la fórmula de actualización 2.1, descrita anteriormente.
     - Establece el estado actual s como el próximo estado s'
3. **Convergencia:** Repite el proceso de exploración y actualización de la tabla Q a lo largo de múltiples episodios hasta que la tabla Q converja a los valores óptimos.

Los parámetros del algoritmo Q-Learning incluyen la tasa de aprendizaje α, que controla la rapidez con la que se actualizan los valores de la tabla Q, y el factor de descuento γ, que determina la importancia relativa de las recompensas futuras en estas actualizaciones. El algoritmo puede terminar después de un número fijo de episodios o cuando la tabla Q haya convergido a valores estables, lo que indica que el agente ha aprendido una política óptima. Estos parámetros y criterios de terminación son cruciales para el ajuste adecuado y la eficacia del algoritmo en la búsqueda de soluciones óptimas en entornos dinámicos.

Zhang et al. (2021) afirman que el algoritmo Q-learning es una herramienta de aprendizaje por refuerzo efectiva, especialmente en entornos desconocidos. No requiere conocer previamente el modelo del ambiente y ofrece ventajas como garantizar la convergencia, tener

menos parámetros y contar con una buena capacidad de exploración. En particular, en el ámbito de toma de decisiones dentro de un entorno desconocido, donde se busca obtener resultados óptimos, el algoritmo Q-learning presenta más beneficios y es objeto de investigación por parte de académicos en la actualidad. En este algoritmo, la 'Q' se refiere específicamente al valor Q (estado S, acción A) de la pareja estado-acción, que indica el rendimiento esperado cuando el agente realiza la acción 'a' (donde 'a' pertenece al conjunto de acciones posibles) bajo el estado 's' (donde 's' pertenece al conjunto de estados posibles). El entorno retroalimentará un valor de recompensa instantáneo correspondiente según el comportamiento del Agente. Por lo tanto, la idea central de este algoritmo es utilizar una tabla Q para almacenar todos los valores Q. En la interacción continua entre el Agente y el entorno, el valor esperado (evaluación) se actualiza a través de la recompensa real obtenida al ejecutar la acción bajo los estados, y al final, se elige la acción que contribuye al máximo retorno.

2.1.5. Aprendizaje adaptativo

El aprendizaje adaptativo, según según Jayasiriwardene and Meedeniya (2023), es una estrategia empleada para diseñar programas que se ajusten a las habilidades, comprensión y requisitos específicos de cada individuo. Este enfoque aprovecha el aprendizaje automático y los algoritmos de inteligencia artificial para supervisar y adaptar el contenido en función del progreso individual, la respuesta a las actividades, los errores y los aciertos, entre otros factores. El propósito principal de esta técnica es fomentar un alto nivel de compromiso y efectividad en el proceso de aprendizaje al personalizar la experiencia según las necesidades y avances de cada individuo, de manera similar a cómo un entrenador personal o un amigo brindaría apoyo y orientación al enseñar a conducir un vehículo por primera vez.

De acuerdo con Hernández et al. (2019), el concepto de aprendizaje adaptativo a menudo se confunde con el aprendizaje personalizado, considerándose ambos como sinónimos; sin embargo, a pesar de compartir similitudes, es crucial distinguir entre ellos. La diferencia principal radica en que el aprendizaje personalizado se limita a la creación y entrega de contenido adaptado a las necesidades específicas de aprendizaje de un individuo en términos de información, área de competencia y contexto. En otras palabras, el contenido se ajusta a las necesidades del usuario, pero el sistema en sí mismo no cambia ni se adapta en respuesta a la interacción del usuario, sino que sigue su curso predefinido. Por lo tanto, se podría argumentar que, aunque el aprendizaje personalizado es similar, en realidad es más general, mientras que el aprendizaje adaptativo lleva esta personalización a un nivel más profundo y preciso, ofreciendo una experiencia de aprendizaje más dinámica y específica para cada individuo.

El aprendizaje adaptativo es según Aberle (2022) parte del aprendizaje interactivo que

aborda las necesidades de las personas a través de vías de aprendizaje, retroalimentación efectiva y recursos complementarios; a diferencia de un plan de estudios único para todos. El avance tecnológico hace que el aprendizaje adaptativo sea más fácil de implementar.

Según la fuente mencionada, hay tres áreas en las que se puede implementar el aprendizaje adaptativo: contenido adaptativo, secuencia adaptativa y evaluación adaptativa: El contenido adaptativo proporciona retroalimentación a la respuesta específica de los estudiantes (por ejemplo, sugerencias, materiales de revisión sobre la habilidad relevante, mayor apoyo) sin cambiar la secuencia general de habilidades; la secuencia adaptativa recopila y analiza continuamente los datos de los estudiantes para cambiar automáticamente lo que un estudiante ve a continuación; la evaluación adaptativa cambia las preguntas que ve un estudiante en función de su respuesta a la pregunta anterior. La dificultad de las preguntas aumentará a medida que el estudiante las responda con precisión, mientras que, si el estudiante tiene dificultades, las preguntas se volverán más fáciles.

2.2. Estado del Arte

Con base en estudios realizados a nivel internacional, se encuentra el artículo de Moghadam et al. (2021) el cual propone que si un agente de pruebas puede aprender la política (forma) óptima para generar la carga de trabajo de prueba para cumplir con un objetivo de prueba, entonces la automatización de pruebas eficiente sería posible sin depender de modelos de sistema o código fuente. En este sentido, se presenta un agente de pruebas de carga auto adaptativo impulsado por el aprendizaje por refuerzo, RELOAD, que aprende la política óptima para la generación de cargas de trabajo de pruebas y genera una carga de trabajo efectiva de manera eficiente para cumplir con el objetivo de la prueba. Una vez que el agente aprende la política óptima, puede reutilizar la política aprendida en actividades de prueba posteriores. Finalmente, los resultados experimentos muestran que el agente de prueba de carga inteligente propuesto puede lograr el objetivo de la prueba con un costo de prueba más bajo en comparación con los procedimientos de prueba de carga comunes y da como resultado una mayor eficiencia de la prueba. entonces sería posible una automatización de pruebas eficiente sin depender de modelos de sistema o código fuente.

El estudio presenta un agente de prueba de carga llamado RELOAD, que utiliza aprendizaje por refuerzo y es autoadaptativo y sin modelo. Este agente aprende a generar cargas de trabajo de prueba de manera efectiva sin necesidad del modelo del sistema o del código fuente, y puede reutilizar la política aprendida en diferentes escenarios de prueba. El proceso de aprendizaje consta de dos fases: inicial y de transferencia. En la fase inicial, el agente aprende la política óptima para generar cargas de trabajo efectivas. Luego, en la fase de

transferencia, puede adaptativamente reutilizar la política aprendida en diferentes escenarios de prueba con distintos objetivos. El agente utiliza el algoritmo Q-learning, que es un método de aprendizaje por refuerzo sin modelo, y una estrategia de selección de acciones adaptativa para poder reutilizar la política aprendida en la fase de transferencia. Además, RELOAD utiliza Apache JMeter como actuador de prueba de carga.

Por otro lado, se encuentra también el estudio de Helali Moghadam (2020) que tiene como objetivo de investigación desarrollar técnicas de aseguramiento del desempeño adaptables y eficientes que cumplan con los objetivos previstos sin acceso a modelos y código fuente. Proponemos tres enfoques que se basan en el aprendizaje sin modelos para resolver los desafíos relacionados con la generación eficiente de casos de prueba de rendimiento, la conservación del rendimiento durante la ejecución (tiempo de respuesta) y la mejora del rendimiento al reducir el makespan (tiempo de finalización). En nuestro estudio, demostramos la eficacia y la capacidad de adaptación de estos enfoques a través de evaluaciones experimentales llevadas a cabo en herramientas prototipo de investigación. Estas herramientas prototipo incluyen entornos de simulación que hemos desarrollado o adaptado para abordar diversos problemas en diferentes áreas de aplicación.

En esta medida, los resultados también señalan que RELOAD, como agente de pruebas de carga impulsado por el aprendizaje por refuerzo adaptativo, descubre de manera efectiva los efectos de diferentes transacciones involucradas en la carga de trabajo y aprende cómo ajustar la carga de transacciones para cumplir con el objetivo de prueba previsto (por ejemplo, alcanzar una cierta tasa de error). El agente de prueba inteligente aprende la estrategia más efectiva para crear una carga de trabajo que cumpla con la tasa de error deseada. Esta estrategia aprendida puede ser reutilizada en futuros escenarios de prueba similares, como las pruebas de carga de regresión en el Sistema en Prueba (SUT). Esto conlleva a una disminución de los gastos asociados con la generación de carga de trabajo en situaciones futuras.

Una contribución importante del estudio antes examinado radica en que el mismo examina como se puede aplicar el aprendizaje por refuerzo RL al rendimiento de sistemas de software, donde también se proponen tres métodos adaptables basados en entradas y sin modelos para manejar los desafíos que identificamos; generación eficiente de casos de prueba para el rendimiento, mantenimiento del rendimiento (tiempo de respuesta) y mejora del rendimiento a la hora de reducir el tiempo que lleva completar una tarea.

Finalmente, se encuentra el estudio de Lizcano et al. (2009) donde se se plantea como objetivo analizar las tendencias en el desarrollo de agentes inteligentes aplicados en la construcción de un aula inteligente. Para lo que se empleó una metodología que consistió en

realizar un proceso de consulta bibliográfica sistemática que ha permitido identificar las tendencias, utilidad y tecnologías de agentes y sistemas multiagente, en el ámbito de procesos de aprendizaje de agentes naturales y artificiales. El artículo muestra la conceptualización fundamental sobre agentes inteligentes, la metodología utilizada para adelantar la revisión, los hallazgos obtenidos en el proceso y la aplicación de los mismos en diferentes aspectos del proyecto Aula Inteligente señalan que existen pocos estudios que aborden la temática, pues generalmente las investigaciones se enmarcan en mostrar la arquitectura del agente y algunos procesos básicos de la construcción del mismo, como algoritmos o técnicas de representación, aprendizaje o comunicación de los agentes. De la misma manera, sobre las técnicas de representación de conocimiento se identifican las redes semánticas y las ontologías como las más utilizadas en los trabajos seleccionados, combinados con procesamiento y búsquedas utilizando minería de datos.

El aporte del articulo antes analizado es la revisión sistemática que se emplea respecto a una serie de aspectos clave relacionados con el desarrollo y documentación de agentes inteligentes, pues si bien no es un estudio aplicativo, si denota un carácter exploratorio que muestra los tipos de arquitectura del agente y ciertos procesos básicos de la construcción del mismo. En esta medida es posible observar algunas proyecciones de la investigación en el ámbito de agentes inteligentes y sistemas multiagente, entre las que se encuentran la personalización y ubicación de recursos q-learning.

Capítulo 3

Desarrollo del Proyecto

El presente proyecto se basó en la aplicación de un agente inteligente impulsado por aprendizaje por refuerzo (RL) aplicado a pruebas de carga. En resumen, el proyecto implicó emplear un agente de prueba de carga denominado RELOAD, el cual detecta los impactos de varias transacciones en la carga de trabajo y aprende a modificarlas para cumplir con el objetivo de rendimiento previsto para un software bajo prueba (SUT).

El propósito principal de este proyecto es identificar el rendimiento de un sistema o aplicación de manera eficiente, que contribuya a garantizar el cumplimiento de los requisitos y detectar problemas de rendimiento, sin necesidad de contar con artefactos como modelos del sistema y código fuente, además de poder reducir el esfuerzo requerido para generar cargas de trabajo a un sistema bajo prueba. Por otro lado la aplicación de reload puede ser beneficiosa por la capacidad del agente de reutilizar la política aprendida en escenarios de prueba similares, lo que puede llegar a reducir costos y tiempo en la generación de cargas de trabajo.

3.1. Agente RELOAD para pruebas de carga

3.1.1. Funcionamiento del agente de pruebas de carga

Como lo indica Moghadam et al. (2021), Reload es un agente de pruebas de carga asistido por aprendizaje por refuerzo (RL), el cual se encarga de aprender la mejor política para generar una carga de trabajo que garantice alcanzar un umbral de tiempos de respuesta o tasa de error predeterminados. El agente tiene la capacidad de aplicar la política que ha aprendido en futuras actividades de prueba, incluso para alcanzar distintos objetivos de prueba. Produce una carga de trabajo para pruebas de manera eficiente, adaptándose para cumplir con los objetivos de prueba sin necesidad de acceder a los modelos del sistema ni al código fuente.

Según el autor mencionado el agente de pruebas de carga Reload basado en aprendizaje por refuerzo (RL), se centra en la observación e interacción con el ambiente o entorno, en este caso el sistema bajo prueba (SUT) inicialmente desconocido para el agente, con el fin

emprender una posible acción que permita detectar y obtener un estado y una señal de recompensa que permita evaluar la eficacia de la acción aplicada para lograr el objetivo, y así, aprender la manera óptioma de tomar decisiones.

El ciclo de aprendizaje por refuerzo entre el agente y el entorno, que deacuerdo con Moghadam et al. (2021), se basa en la interacción del agente con el entorno, que se define mediante estados, acciones y recompensas como se muestra en la Figura 3.1.

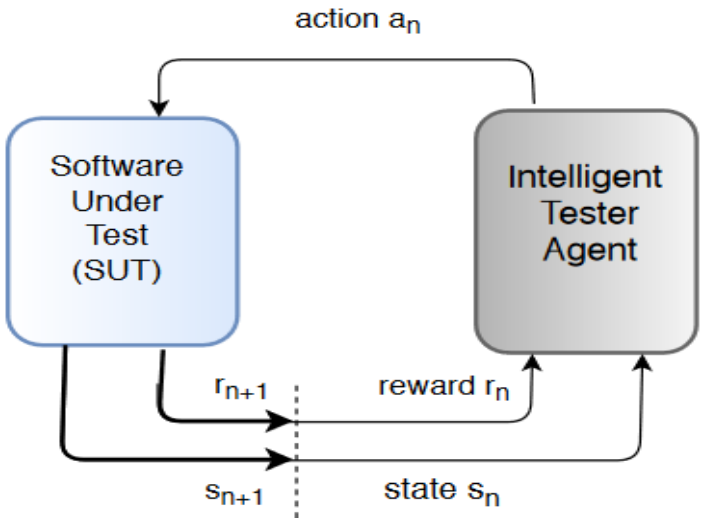


Figura 3.1: Ciclo de aprendizaje por refuerzo entre el agente y el entorno. Tomada de Helali Moghadam (2020)

- **Acciones**: Las acciones se definen como ajustes en la carga de transacciones constituyentes en una carga de trabajo, en términos del número de usuarios virtuales ejecutando cada transacción. Estas acciones implican modificar la carga de transacciones para optimizar el rendimiento del sistema bajo prueba. El agente de prueba toma una acción en cada paso de aprendizaje después de detectar el estado del sistema bajo prueba.

- **Estados**: Los estados se definen como las situaciones actuales del sistema bajo prueba (SUT) y representan la información que el agente de prueba tiene en un momento dado. Estos estados son fundamentales para que el agente tome decisiones sobre las acciones a realizar. En cada paso del aprendizaje, el agente detecta el estado actual del SUT, selecciona una acción basada en este estado y luego ejecuta la acción ajustando la carga de transacciones en la carga de trabajo. Después de ejecutar la acción, el agente identifica el nuevo estado del SUT y calcula la recompensa recibida.

- **Recompensas**: Las recompensas son señales de retroalimentación que recibe el agente de prueba después de tomar una acción en el sistema bajo prueba (SUT). Estas recompensas indican qué tan efectiva fue la acción tomada por el agente para alcanzar el objetivo de la prueba.

El agente opera siguiendo una política, que establece una relación entre los estados del entorno y las acciones que debe tomar. La política óptima es aquella que, en conjunto, maximiza la recompensa esperada a lo largo del tiempo. El agente logra aprender esta política óptima mediante un proceso iterativo en el que evalúa y ajusta su comportamiento en función de las recompensas recibidas, buscando constantemente mejorar su desempeño. La selección de acciones se basa en una estrategia de exploración y explotación, donde el agente puede probar diferentes acciones o confiar en la política aprendida para seleccionar acciones de alto valor.

En esta medida, Moghadam et al. (2021) define que el agente de prueba RELOAD se basa en Q-learning, un algoritmo de aprendizaje por refuerzo sin modelos, como su principal técnica de aprendizaje. Q-learning se encarga de aprender una función de valor óptimo, conocida como función de acción-valor $Q^*(s,a)$, a partir de la cual se puede derivar la política óptima. Esta función de acción-valor óptima proporciona una estimación del rendimiento esperado a largo plazo dado un estado específico s y una acción a, seguida por la aplicación de la política óptima. La política óptima, por su parte, elige la acción que maximiza dicho rendimiento esperado desde el estado s.

Según el autor mencionado, el agente de pruebas asume dos fases de aprendizaje, la fase inicial de aprendizaje y la fase de transferencia. Durante esta etapa inicial, el agente de prueba RELOAD se dedica a aprender la política óptima para generar una carga de trabajo eficaz y alcanzar los objetivos de la prueba. Empleando Q-learning como técnica de aprendizaje por refuerzo, para iterativamente aprender la política que maximiza la recompensa acumulada a lo largo del tiempo.

3.1.2. Parámetros y características del agente para pruebas de carga

Según Moghadam et al. (2021), los principales componentes del agente RELOAD son la detección del estado, la aplicación de acciones y el cálculo de la recompensa, a continuación de describen con mas detalle.

- **Detección del estado:** El tiempo promedio de respuesta y la tasa de error, dos medidas de rendimiento fundamentales para el agente, se emplean para evaluar el desempeño del sistema bajo Prueba (SUT). Estas medidas se clasifican en distintas

clases discretas, denominadas Bajo, Normal y Alto para el tiempo de respuesta, y Baja y Alta para la tasa de error como se muestra en la Figura 3.2. La combinación de estas clases conforman las categorías discretas que describen el estado del sistema. El agente adquiere estas métricas del actuador de prueba en cada paso de aprendizaje, lo que le permite identificar el estado actual del SUT.

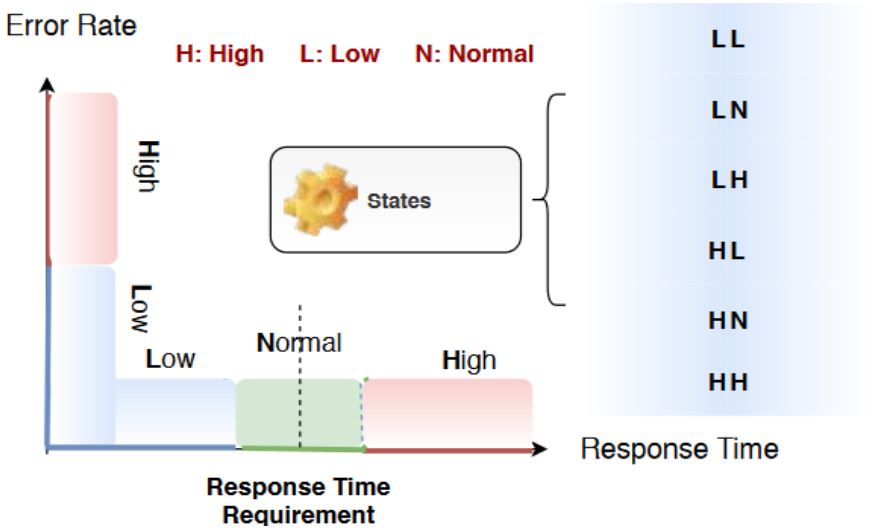


Figura 3.2: Estados del SUT. Tomada de Helali Moghadam (2020)

- **Acciones:** En cada uno de los pasos del aprendizaje, el agente de pruebas ejecuta una acción después de identificar el estado del Sistema bajo Prueba (SUT). Estas acciones se definen como la modificación de la carga de las transacciones que componen la carga de trabajo, expresada en términos del número de usuarios virtuales que ejecutan cada transacción. La Tabla 3.1 de la sección 3.2.1 lista el conjunto de transacciones del sistema sujeto a pruebas. Estas se describen a detalle en la sección 3.2 de este documento. En cada transacción de la carga de trabajo, se deben tener en cuenta cada función específica junto con sus dependencias funcionales asociadas. En consecuencia, se considera cada función en una transacción junto con sus dependencias funcionales correspondientes. En la ecuación 3.1 tomada de Helali Moghadam (2020), se muestra el conjunto de acciones para el agente.

$$
\begin{aligned}
ActionList = \{Uaction_k, 1 \leq k \leq |List\,of\,Transactions|\} \\
action_k = \{W_n^{T_j} = W_{n-1}^{T_j}, forj \neq k, \\
W_n^{T_j} = W_{n-1}^{T_j} + \frac{W_{n-1}^{T_j}}{3}, forj = k, \\
T_j \in List\,of\,Transactions, \\
1 \leq j \leq |List\,of\,Transactions|\}
\end{aligned}
\tag{3.1}
$$

Donde T_j representa una transacción específica del Sistema bajo Prueba. ${W_n}^{TJ}$ se refiere a la carga de la transacción T_j en el paso de tiempo n, es decir, la cantidad de usuarios que están ejecutando esta transacción en ese momento. Después de que el agente ha tomado una decisión sobre qué acción llevar a cabo, se procede a generar un plan de pruebas basado en esa decisión y se ejecuta en el SUT utilizando el actuador de pruebas designado, que en este caso es Apache JMeter.

- **Recompensa:** Después de ejecutar la acción seleccionada y ajustar la carga de trabajo según corresponda, el agente de pruebas recibe una señal de recompensa que refleja la efectividad de la acción realizada para avanzar hacia el objetivo de la prueba. Según Helali Moghadam (2020), se define la siguiente función para representar esta señal de recompensa:

$$
R_n = (\frac{RT_n}{RT_{threshold}})^2 + (\frac{ER_n}{ER_{threshold}})^2 \tag{3.2}
$$

Donde R_n denota la recompensa, RT_n el tiempo medio de respuesta y ER_n la tasa promedio de error en el paso n (Cada paso). Además, $RT_{threshold}$ establece el tiempo de respuesta y $ER_{threshold}$ la tasa de error, ambos establecidos para el objetivo de la prueba.

De acuerdo con el autor, en el aprendizaje por refuerzo (RL), el objetivo del agente es aprender la mejor manera de actuar en un entorno para lograr su objetivo. Esto se logra mediante un proceso iterativo de evaluación y mejora de la política en cada paso del aprendizaje. El agente evalúa su desempeño actual y luego busca maneras de mejorar a través de ε-greedy (enfoque codicioso) para maximizar sus recompensas a largo plazo y finalmente

converger en la política óptima.

En la RL sin modelos, el agente puede aprender directamente una política de acciones óptima o indirectamente a través de la función de valor óptimo *Q*(s,a)*. En el algoritmo Q-learning, por ejemplo, el agente calcula la función de valor óptimo para guiar sus acciones hacia la política más beneficiosa, esto proporciona el rendimiento esperado a largo plazo, dado el estado s, tomando una acción arbitraria a, y luego siguiendo la política óptima. Según Helali Moghadam (2020), la función de acción valor se representa en la ecuación 3.3.

$$\begin{gathered} Q^*(s,a) = argmax\, E^{\pi}[q_n | s_n = s, a_n = a] \\ q_n = \sum_{k=0}^{\infty} \gamma^k R_{n+k+1} \end{gathered} \tag{3.3}$$

En la definición, γ representa un factor de descuento aplicado a las recompensas futuras, mientras que q_n representa el rendimiento acumulado a largo plazo. En términos generales, la política óptima busca maximizar el rendimiento esperado partiendo desde un estado dado s. Además, según la definición de la función *Q*(s, a)*, dada *Q**, la acción óptima para el estado *s, a*(s)*, se obtiene como se muestra en la ecuación 3.4.

$$a^*(s) = argmax\, Q^*(s, a') \tag{3.4}$$

Con el fin de alcanzar la política óptima, los valores Q son registrados en una tabla Q y se tiene en cuenta la experiencia acumulada por el agente. Durante el proceso de aprendizaje, los valores Q son ajustados gradualmente según la ecuación 3.5, tomada de Helali Moghadam (2020).

$$Q(s_n, a_n) = (1 - \alpha) Q(s_n, a_n) + \alpha[R_{n+1} + \gamma max_{a'} Q(s_{n+1}, a')] \tag{3.5}$$

Donde α está comprendido entre 0 y 1 y se utiliza para ajustar la tasa de aprendizaje del agente, controlando así el impacto que tienen los nuevos valores Q sobre los anteriores. Reload plantea generar una carga de trabajo eficaz para alcanzar un objetivo de prueba previsto, esto se asemeja a un problema secuencial de toma de decisiones. El uso de aprendizaje por refuerzo sin modelos es una solución beneficiosa para este objetivo y tipo de problema, dado que tanto el entorno del sistema bajo prueba (SUT) como la plataforma de ejecución

son inicialmente desconocidos para el agente de prueba. Por lo tanto, el agente aprende y adapta una política óptima para generar una carga de trabajo eficaz que cumpla con el objetivo de la prueba.

El aprendizaje por refuerzo, ε-greedy es un método ampliamente reconocido para la selección de acciones cuando se busca la política óptima en un problema de toma de decisiones. Este método asegura una exploración suficiente y continua necesaria para encontrar la mejor política, al mismo tiempo que equilibra adecuadamente la exploración del espacio de estados y acciones con la explotación de la función de valor aprendida, ajustando el valor de ε se ajusta el grado de exploración frente a la explotación, lo que permite al agente seleccionar una acción con alta probabilidad basada en la función de valor aprendida (1-ε) o una acción aleatoria con probabilidad ε en un estado dado.

Según Helali Moghadam (2020), los siguientes algoritmos detallan el procedimiento de aprendizaje en el agente de pruebas de carga basado en aprendizaje por refuerzo propuesto.

Algorithm 1 Pruebas de carga basadas en RL

Require: S, A, α, γ;
Inicializar valores Q,
$Q(s,a) = 0\ \forall s \in S, \forall a \in A\ and\ \varepsilon = u, 0 < u < 1$;
while Not (convergencia inicial alcanzada) **do**
Episodio Aprendizaje(con estrategia inicial de selección de acciones, por ejemplo ε-greedy, ε inicializado);
end while
Almacenada la política aprendida;
Adapta la estrategia de selección de acciones al aprendizaje por transferencia, es decir, ajusta ε en ε-greedy;
while verdadero **do**
Episodio Aprendizaje(con estrategia inicial de selección de acciones, por ejemplo, nuevo valor de ε);
end while

Algorithm 2 Episodio de aprendizaje

repeat

1. Detectar el estado (S_n) del SUT;
2. Seleccionar una acción (Ecuación 3.1) teniendo en cuenta la estrategia, por ejemplo ε-greedy: seleccionar $a_n = argmax_{a \in A} Q(s_n, a)$ con probabilidad (1-ε);
3. Realizar la acción seleccionada: Ajusta la carga de trabajo y ejecuta la carga modificada en el SUT;
4. Detectar el nuevo estado (S_{n+1}) del SUT;
5. Calcula la recompensa, R_{n+1};
6. Actualiza el valor Q del par de estado anterior y acción realizada $Q(s_n, a_n) = (1 - \alpha)Q(s_n, a_n) + \alpha[R_{n+1} + \gamma max_{a'} Q(s_{n+1}, a')]$

until que se cumpla el total de episodios o se cumpla con el objetivo de la prueba;

3.1.3. Arquitectura del agente para pruebas de carga

Segun Helali Moghadam (2020), el agente inteligente de pruebas de carga RELOAD emplea una herramienta para generar carga, en este caso Apache JMeter, para llevar a cabo la ejecución de la carga de trabajo a un Sistema bajo Prueba (SUT). JMeter permite generar estas cargas mediante usuarios virtuales o hilos de ejecución, que simulan la cantidad de usuarios reales que pueden llegar a interactuar con el sistema, además provee métricas de rendimiento de los resultados de las pruebas de carga. Los datos como tiempos de respuesta y tasa de error que provee JMeter son la base para detectar el estado del SUT y permite que el agente tome la mejor acción. En cada ciclo de aprendizaje el agente de pruebas sigue los siguientes pasos:

1. Detecta el estado actual del Sistema bajo Prueba (SUT) mediante métricas como el tiempo de respuesta y la tasa de error.

2. Selecciona una acción, que implica ajustar la carga de las transacciones que conforman la carga de trabajo.

3. Ejecuta la acción y recibe una recompensa que indica la eficacia de la acción para alcanzar los objetivos de la prueba.

4. Actualiza la función de valor Q, que representa la política óptima, basándose en la recompensa obtenida.

A través de este proceso iterativo de evaluación y mejora de la política, el agente aprende a generar cargas de trabajo cada vez más efectivas.

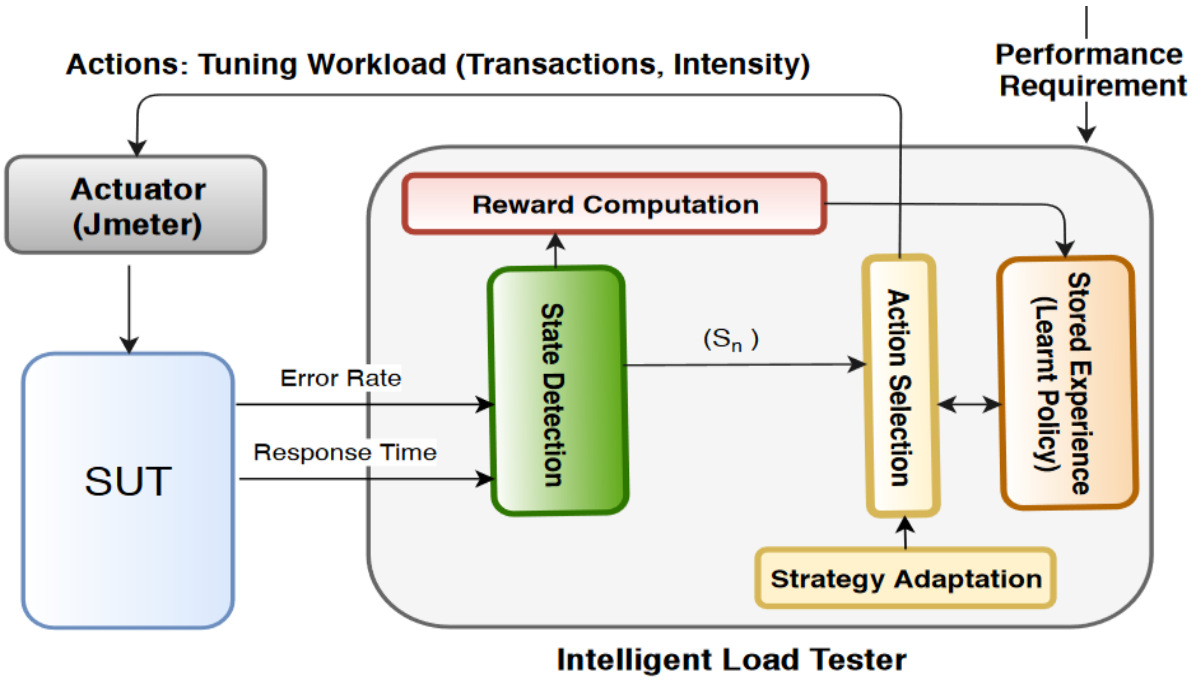


Figura 3.3: Arquitectura de RELOAD. Tomada de Helali Moghadam (2020)

Después de alcanzar la convergencia en la fase inicial, el agente de pruebas RELOAD pasa a la etapa de transferencia de aprendizaje. En esta fase, el agente utiliza la política previamente aprendida en nuevos escenarios de prueba similares, como cambios en los objetivos de la prueba, además el agente continúa aprendiendo durante esta fase para mantener actualizada la política, aunque se apoya principalmente en la política previamente aprendida.

3.2. Definición del entorno del agente de pruebas

El entorno se compone por una maquina virtual (VM) de uso general desplegada en la plataforma de Microsoft Azure que alberga el sistema sujeto a prueba (SUT), esta VM cuenta con 2 vCPU a 2.3 GHz y 8 GB de memoria RAM y utiliza el sistema operativo Windows 10 como base. Una maquina con sistema operativo Windows 11 con 4 núcleos a 4.3 GHz y 16 GB de memoria RAM, donde se aloja el agente inteligente RELOAD y el actuador para pruebas de carga JMeter 5.6.3.

3.2.1. Sistema Sujeto a Pruebas (SUT)

Para la definición del entorno en el que interactúa el agente de pruebas, se definió una apliación web en desarrollo llamada CycleConnect (ver la figura 3.4), enfocada a ciclistas, en la que permite gestionar procesos referentes a usuarios, bicicletas, rutas, eventos, promoción de eventos y noticias, desarrollada en las plataformas de React Native, Node.js y MongoDB.

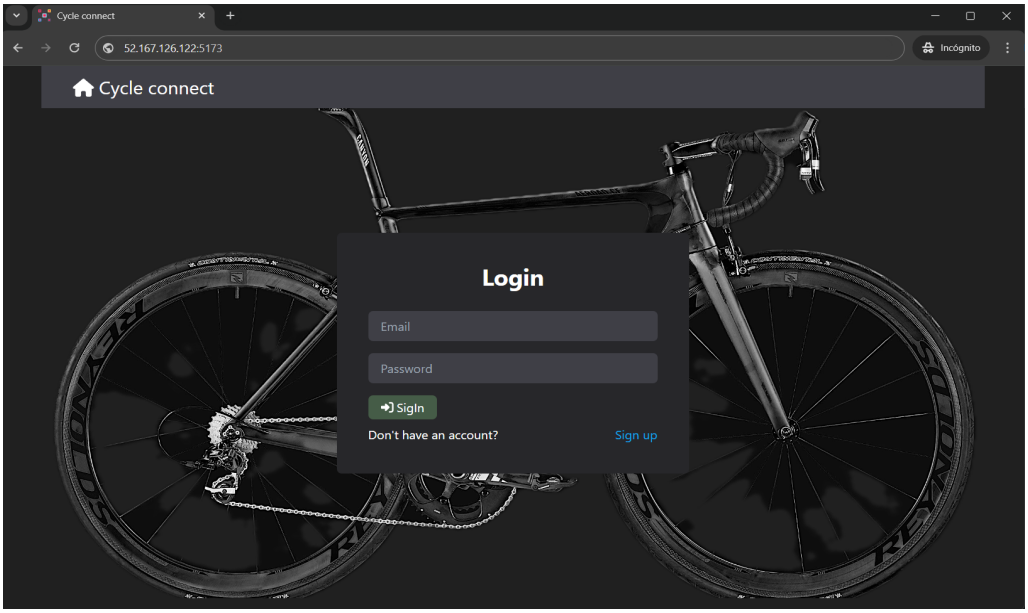


Figura 3.4: Inicio de sesión del SUT.

Las funcionalidades y transacciones de la aplicación web se presentan a continuación en la tabla 3.1 y pueden visualizarse en detalle en la en la sección 6.1 de los anexos.

Tabla 3.1: Listado de transacciones del SUT.

Transacción	Descripción
Inicio	Acceso a la página de inicio
Registro de usuario	Registrar y añadir usuarios
Inicio de sesión	Iniciar sesión en la aplicación
Registro de bicicletas	Registrar y añadir bicicletas
Obtener bicicletas	Obtener bicicletas por usuario
Registro de rutas	Registrar y añadir rutas
Registro de eventos	Registrar y añadir eventos
Registro de participantes	Registrar participantes a eventos
Cerrar sesión	Cerrar la sesión del usuario

3.2.2. Definición del plan de pruebas de carga en JMeter

Para comenzar, fue necesario instalar Apache JMeter. Para esto se descargó la versión 5.6.3 de JMeter desde el sitio web oficial, siendo esta la última versión estable disponible. Antes de comenzar a configurar las pruebas en JMeter, fue fundamental realizar una evaluación exhaustiva del Sistema bajo Prueba (SUT), en este caso, la aplicación CycleConnect. Esto implicó comprender en detalle las funcionalidades de la aplicación, el entorno de implementación y cualquier aspecto relevante que puediera afectar su rendimiento. Posteriomente se procedió a realizar los siguientes pasos:

- **Identificación de escenarios de prueba:** Durante la evaluación del SUT, fue crucial identificar las transacciones más importantes que serán objeto de las pruebas de carga e identificar sus dependencias funcionales, puntos de acceso (endpoints), así como los métodos y la información requerida para los mismos, como se puede evidenciar en la tabla 3.2. Estas transacciones representan las acciones principales que los usuarios realizarán en la aplicación y, por lo tanto, son fundamentales para garantizar su funcionamiento óptimo bajo diferentes cargas de trabajo.

Tabla 3.2: Listado de endpoints por transacción.

Transacción	Punto de acceso	Método
Registro de usuario	/api/v1/user	POST
Inicio de sesión	/api/v1/auth/login	POST
Registro de bicicletas	/api/v1/bicycle	POST
Obtener bicicletas	/api/v1/bicycles	GET
Registro de rutas	/api/v1/route	POST
Registro de eventos	/api/v1/event	POST
Registro de participantes	/api/v1/users-event	POST
Cerrar sesión	/api/v1/auth/logout	GET

- **Creación del plan de pruebas:** Se creó un plan de pruebas en JMeter, que servirá como contenedor para todas las configuraciones y pruebas relacionadas con CycleConnect, como se muestra en la figura 3.5.

- **Agregar grupos de usuarios:** Se crearon los grupos de usuarios en JMeter para simular el comportamiento de los usuarios reales de CycleConnect. Para esto se incluyeron las diferentes transacciones con sus dependencias funcionales, por ejemplo, la transacción Registro de bicicletas, tiene dependencias con Inicio de sesión y ésta a su vez, con que exista un usuario registrado en el sistema.

- **Agregar elementos de preba:** Se añadieron elementos de prueba, como HTTP Request con la estructura JSON e información necesaria, para cada una de las transacciones identificadas anteriormente, HTTP Header Manager para controlar las cabeceras de cada una de las peticiones y configuraciones como Random Variable para generar datos aleatorios y Regular Expression Extractor para almacenar y compartir información de respuestas entre las diferentes transacciones. Estos elementos simulan las interacciones de los usuarios con la aplicación CycleConnect mediante el protocolo HTTP.

- **Configuración de cargas de trabajo:** Para cada transacción, se configuraron las cargas de trabajo en JMeter. Esto incluye especificar el número de usuarios concurrentes, la frecuencia de las solicitudes y otras características relevantes para simular un uso realista de la aplicación.

- **Agregar informes**: Se agregaron informes para supervisar el rendimiento de la aplicación durante las pruebas de carga y analizar los resultados de las pruebas.

- **Ejecución de las pruebas:** Se ejecutó el plan de pruebas utilizando JMeter, utilizando una carga mínima para cada una de las transacciones, con el fin asegurarse que el plan de pruebas es ideal para simular el comportamiento de los usuarios en la aplicación CycleConnect y analizar los resultados. Cada una de las transacciones del plan de pruebas se incluye en la sección 6.2 de los anexos.

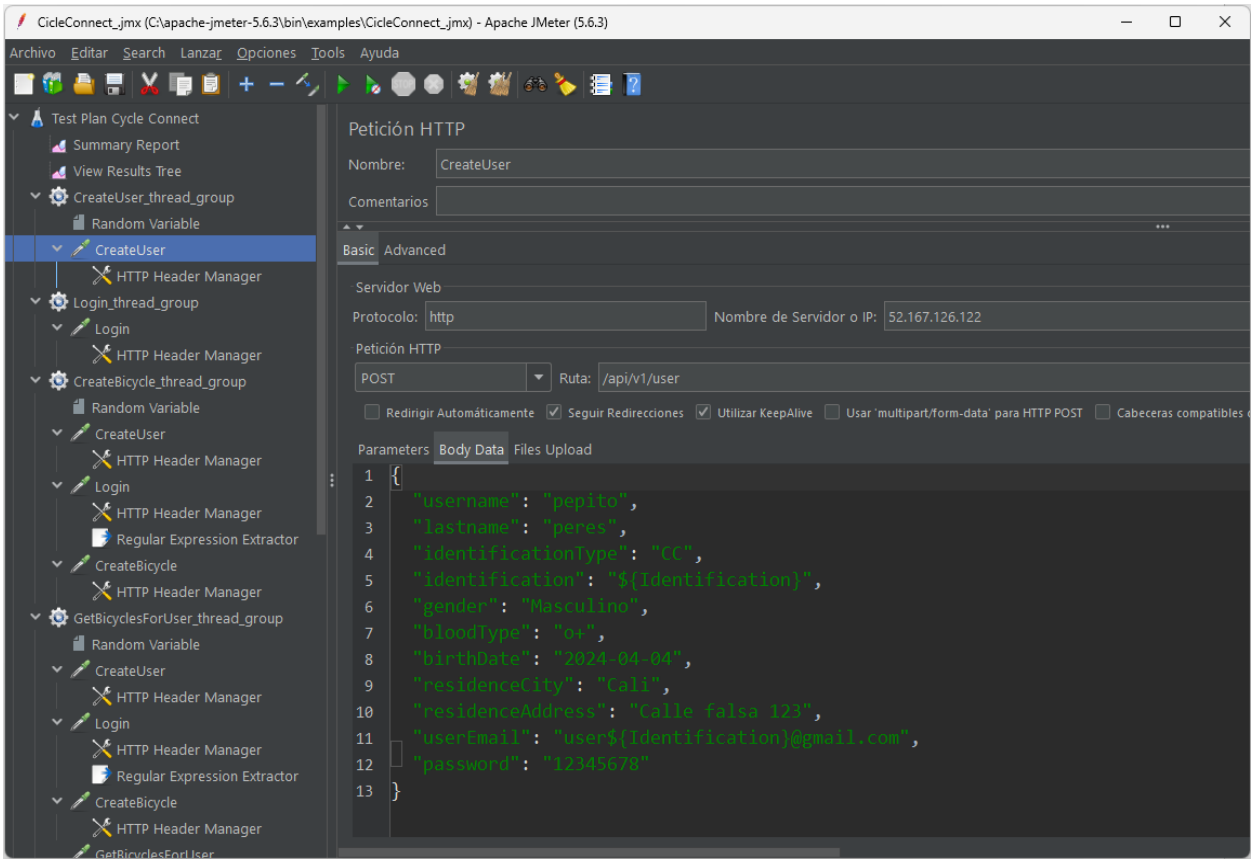


Figura 3.5: Plan de pruebas CycleConnect.

3.2.3. Configuración del agente de pruebas de carga RELOAD

Para la configuración del agente de pruebas RELOAD, primero se realizó la instalación de las herramientas necesarias como GIT en la versión 2.42.0 para el control de versiones e Intellij IDEA Community Edition 2023.3.1 como entorno de desarrollo, esta última ya que el agente esta desarrollado bajo la plataforma de Java, permitiendo una buena integración.

Posteriormente se clonó el proyecto Deeplearning4j (DL4J) de código abierto [1], este contiene una serie de proyectos y librerías destinadas a respaldar las necesidades de aprendizaje profundo basado en la máquina virtual de java (JVM). Para la configuración del agente de pruebas de carga, se utilizó el proyecto específico rl4j-examples, que soporta las librerías de aprendizaje por refuerzo requeridas para poder alojar el agente de pruebas de carga. Para esto se realizó un análisis e investigación de las diferentes versiones del proyecto y poder seleccionar la adecuada, para este caso se utilizó la versión de la fecha 27/10/2020 específicamente la confirmación de código (commit) 903d316e, esto con el fin de tener una relación con las fechas en que se publicó el agente y no tener conflictos con las librerías utilizadas.

Seguido a lo anterior, se agregaron los archivos del agente RELOAD en su versión 0.2 [2] al proyecto anteriormente descrito y la importación de las dependencias requeridas, incluyendo la de Apache Jmeter 5.6.3.

Finalmente, se realizó un análisis del código fuente del agente de pruebas, para poder identificar los principales aspectos configurables e incluir las diferentes transacciones del sistema sujeto a pruebas (SUT), el plan de pruebas previamente creado en Apache JMeter, los grupos de usuarios definidos y las configuraciones respectivas de la herramienta, como rutas para los archivos de registro de salida (logs), ubicación de la herramienta Apache JMeter y su archivo de propiedades. Además de esto, los valores de los objetivos de la prueba como tiempo de respuesta máximo y máxima tasa de error, cantidad de episodios máximos para la ejecución de las pruebas y los hiperparámetros que no son aprendidos directamente por el agente como la tasa de aprendizaje y el factor de descuento.

En la tabla 3.3, se listan las principales clases del agente de pruebas basado en aprendizaje por refuerzo y Q-Learning, donde se explica de manera breve su propósito y configuración.

[1] https://github.com/deeplearning4j/deeplearning4j-examples
[2] https://github.com/mahshidhelali/RL-Assisted-Performance-Testing

Tabla 3.3: Clases principales del agente de pruebas (Q-Learning).

Clase	Propósito	Configuración
ApplyAproach	Clase principal para ejecutar el agente.	Máximo tiempo de respuesta, Máxima tasa de error, Máximo número de episodios, ruta final para los archivos de registro (Logs), tasa de aprendizaje y factor de descuento.
QLearning	Controlar los episodios de aprendizaje en cada paso de la ejecución, detectar el estado del SUT, seleccionar la acción a realizar y calcular la recompensa.	Establecer la cantidad de transacciones configuradas en el SUT y definir epsilon que por defecto se establece en una reducción a lo largo del tiempo (decaying epsilon).
SUT	Modificar la carga de cada transacción, ejecutar la prueba y obtener la carga de trabajo total para cada ejecución.	Se definen todas las transacciones que se van a ejecutar en el plan de pruebas, asi como se llamaron en la herramienta JMeter y la carga inicial por transacción.
LoadTester	Inicializa las configuraciones, ejecuta las transacciones y el plan de pruebas con ayuda de la herramienta JMeter.	Se incluyen las rutas donde se encuentra JMeter, el archivo de propiedades y el plan de pruebas generado en JMeter.
CsvWriter	Generar los archivos CSV de las transacciones realizadas por el agente y generar la tabla Q.	Se incluyen las transacciones del sistema sujeto a pruebas (SUT) para visualizar en los archivos de registro.

3.3. Ejecución de las pruebas de carga mediante el agente

En esta sección, se detalla la ejecución de las pruebas utilizando el agente inteligente Reload para evaluar el rendimiento del sistema sujeto a prueba. Inicialmente, se definieron una serie de experimentos diseñados para identificar la configuración óptima del agente, que

se detallan en la sección 4.1. Con base en los resultados obtenidos, se seleccionó la mejor configuración y se procedió a la fase de aprendizaje, donde el agente utilizó técnicas de aprendizaje por refuerzo para perfeccionar su política de prueba. Posteriormente, se implementó la fase de transferencia, donde el agente utilizó la política aprendida con un objetivo diferente, para verificar su efectividad.

3.3.1. Definir los experimentos para evaluar la eficiencia del agente de pruebas

Para la ejecución de las pruebas de carga con el agente, se estableció el objetivo de la prueba para alcanzar un estado de rendimiento donde el tiempo de respuesta del sistema sujeto a pruebas (SUT) supere los 1500 milisegundos o la tasa de error de las transacciones sea mayor al 10 %, además se realizó un muestreo de manera empírica teniendo en cuenta la configuración de los hiperparámetros de tasa de aprendizaje (alpha - α), factor de descuento (gamma - γ) y la exploración-explotación (epsilon - ε), estas configuraciones se describen en la tabla 4.1 de la sección 4.1.

3.3.2. Ejecución del agente en la fase de aprendizaje

Para la definición de las muestras experimentales e identificar la mejor configuración para realizar las pruebas, se analizó el comportamiento del agente cambiando un parámetro, mientras los otros se mantenian constantes, para este caso se utilizaron los valores de referencia de $\alpha = 0.5$, $\gamma = 0.5$ y $\varepsilon =$ decaying.

Estas muestras experimentales se ejecutaron el mismo número de veces, en este caso el mismo número de episodios para cada configuración. El agente de pruebas de carga define para cada episodio de aprendizaje una secuencia completa de estados y acciones hasta alcanzar el objetivo de la prueba, actualizando la política a lo lardo de cada episodio. Además, se realizó una prueba de carga de linea base estándar, que aplica una carga de trabajo que contiene todas las transacciones con el mismo número de usuarios virtuales y posteriormente aumenta el número de usuarios en pasos fijos de un 33 % hasta cumplir con el objetivo de la prueba, lo anterior y los resultados de los experimentos se describen a detalle en la sección 4.2.

A continuacion se presentan los resultados correspondientes a la fase de aprendizaje inicial. Durante un total de 40 episodios el agente ejecuto las transacciones ajustando la carga de trabajo en cada uno de los pasos de cada episodio, buscando la convergencia de la política aprendida, que se produce después de unos 32 episodios para los valores de referencia de α = 0.5, $\gamma = 0.5$ y $\varepsilon =$ decaying. En la figura 3.6 se muestra el número de usuarios virtuales

generados para producir la carga de trabajo necesaria para cumplir con el objetivo de la prueba y en la tabla 3.4 se muestra el resultado generado por el agente para la tabla Q, donde se puede evidenciar los objetivos de la prueba, los parámetros del agente, el número total de episodios, la carga de trabajo en cada episodio (usuarios virtuales), el paso en el que alcanzó el objetivo para cada episodio, el tiempo de respuesta y la tasa de error.

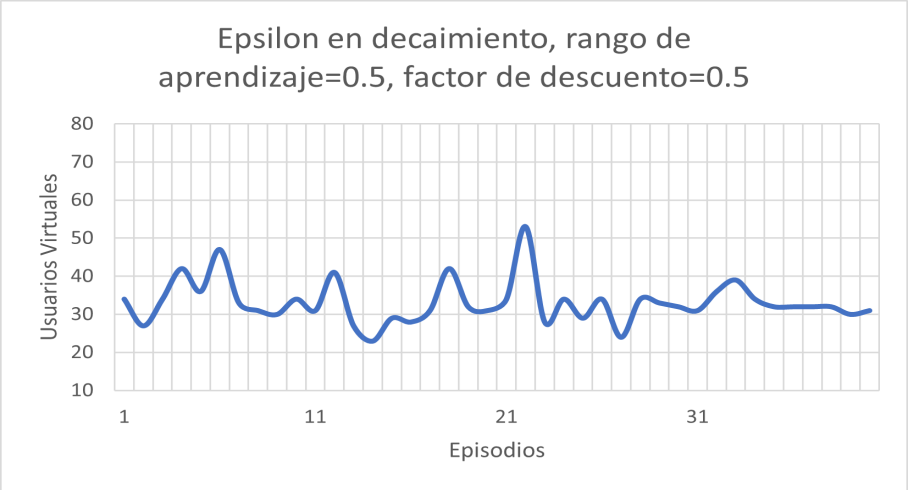


Figura 3.6: Agente con Q-Learning $\alpha = 0.5$, $\gamma = 0.5$ y $\varepsilon =$ decaying.

La tabla 3.4 detalla los resultados de las pruebas de carga ejecutadas utilizando la configuración más eficiente del agente inteligente Reload, basado en Q-Learning, donde se puede evidenciar la cantidad total de episodios ejecutados para identificar las cargas de trabajo necesarias y los tiempos de respuestas y tasas de error en cada episodio.

Tabla 3.4: Tabla Q, $\alpha = 0.5$, $\gamma = 0.5$ y $\varepsilon =$ decaying.

Tiempo max. respuesta	**Tasa max. error**	**Retraso episodio**	**Rango aprendizaje**	**Factor descuento**
1500	0.1	2	0.5	0.5
Episodio	**Carga de trabajo**	**Paso episodio**	**Tiempo respuesta**	**Tasa error**
1	34	14	2047.6326530612246	0.0
2	27	9	1529.972602739726	0.0
3	34	12	1958.366972477064	0.0

Continúa en la siguiente página

Tabla 3.4 – continúa desde la página anterior

Episode	TotalWorkload	EpisodeStep	ResponseTime	ErrorRate
4	42	20	1549.6363636363637	0.0
5	36	17	1770.978021978022	0.0
6	47	21	1884.1203703703704	0.0
7	33	13	1851.8865979381444	0.0
8	31	12	1727.7674418604652	0.0
9	30	12	1515.6913580246915	0.0
10	34	16	2087.4886363636365	0.0
11	31	13	1628.6125	0.0
12	41	19	1507.7727272727273	0.0
13	27	8	1571.871794871795	0.0
14	23	5	1509.0338983050847	0.0
15	29	10	1712.2048192771085	0.0
16	28	10	1533.7972972972973	0.0
17	31	11	1941.888888888889	0.0
18	42	18	1515.188679245283	0.0
19	32	13	1934.4777777777779	0.0
20	31	11	1537.0210526315789	0.0
21	34	12	2011.6078431372548	0.0
22	53	23	1605.8938053097345	0.0
23	28	10	1527.7368421052631	0.0
24	34	14	1680.9878048780488	0.0
25	29	9	1665.8076923076924	0.0
26	34	13	1746.5463917525774	0.0
27	24	6	1621.1029411764705	0.0
28	34	12	1522.050505050505	0.0
29	33	10	1787.5408163265306	0.0
30	32	12	1988.89010989011	0.0
31	31	11	1884.9878048780488	0.0
32	36	13	1710.8039215686274	0.0
33	39	16	1559.8648648648648	0.0
34	34	11	1586.1057692307693	0.0
35	32	10	1525.030612244898	0.0
36	32	10	1525.41	0.0
37	32	10	1500.4782608695652	0.0
38	32	10	1857.6224489795918	0.0

Continúa en la siguiente página

Tabla 3.4 – continúa desde la página anterior

Episode	TotalWorkload	EpisodeStep	ResponseTime	ErrorRate
39	30	8	2017.891304347826	0.0
40	31	9	1562.1030927835052	0.0

La importancia de esta tabla Q radica en su papel fundamental para el funcionamiento del agente Reload. La tabla almacena los valores para cada par estado-acción, permitiendo al agente tomar decisiones informadas sobre qué acciones ejecutar en cada estado para maximizar las recompensas futuras. Al actualizar continuamente estos valores en función de los objetivos de la prueba, el agente aprende a identificar y aplicar las cargas de trabajo que optimizan el cumplimiento de dichos objetivos.

La tabla Q generada no solo refleja el aprendizaje y la adaptación del agente durante las pruebas, sino que también proporciona una herramienta crítica para evaluar y mejorar continuamente la eficacia de las pruebas de carga. Esta capacidad de adaptarse y optimizarse según el entorno de ejecución subraya la ventaja significativa de utilizar un enfoque basado en aprendizaje por refuerzo para pruebas de rendimiento. La tabla 3.4, muestra los diferentes episodios y cargas de trabajo generados para cumplir el objetivo, evidenciando un variabilidad importante en los primeros episodios y observando una estabilidad entre las cargas generadas después de los 32 escenarios, donde la variabilidad disminuye. Esta tabla es la base para generar la gráfrica de rendimiento expresada en usuarios virtuales (cargas de trabajo) y número de episodios (figura 3.6). La convergencia observada en los episodios finales destaca la efectividad del agente en desarrollar una política óptima de generación de cargas de trabajo.

3.3.3. Ejecución del agente en la fase de aprendizaje por transferencia

Durante el aprendizaje por transferencia, es decir, cuando al agente reutiliza la política aprendida en situaciones o escenarios de prueba similares, tras la convergencia inicial de 40 episodios en total, se continuó con 10 episodios adicionales (episodios 41 a 50), manteniendo epsilon bajo para llevar al agente a confiar en la política aprendida y cambiando el objetivo de la prueba a lo largo de los episodios de aprendizaje por transferencia de manera gradual, realizando incrementos de 100 milisegundos para un tiempo de respuesta máximo de 2500 milisegundos y 0.01 para una tasa de error máxima de 0.2, es decir del 20 %. En la figura 3.7 se muestra el número de usuarios virtuales generados para producir la carga de trabajo necesaria para cumplir con el nuevo objetivo de la prueba y en la tabla 3.5 se muestra el resultado

generado por el agente para la tabla Q, que es la base para evaluar el comportamiento del mismo.

Tabla 3.5: Tabla Q, aprendizaje por transferencia

Episodio	Carga de trabajo	Paso episodio	Tiempo respuesta	Tasa error
1	58	12	1851.0	0.0
2	79	17	1868.458904109589	0.0
3	71	11	2154.1954022988507	0.0
4	83	14	1947.0169491525423	0.0
5	76	13	2179.7684210526318	0.0
6	83	13	2163.8133333333335	0.0
7	120	20	2248.6301886792453	0.0
8	137	19	2318.017142857143	0.0
9	139	21	2444.769230769231	0.0
10	142	24	2550.6943005181347	0.0

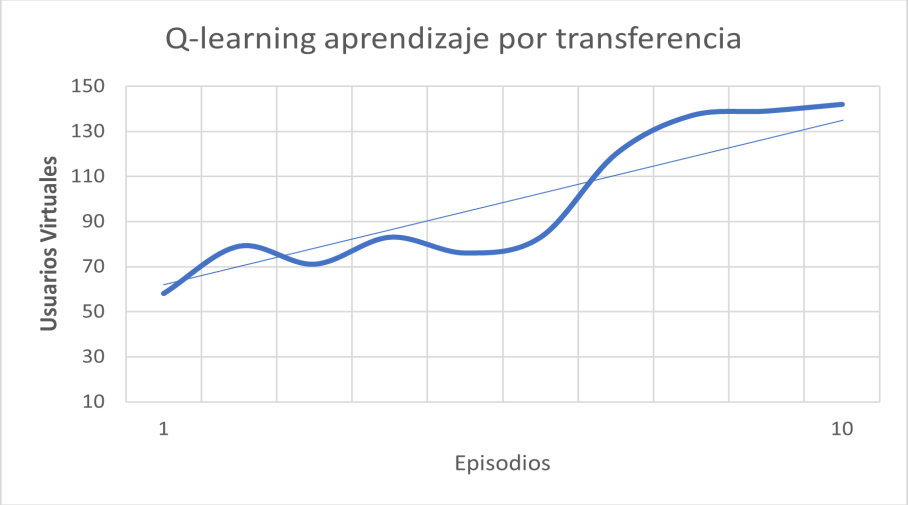


Figura 3.7: Agente en aprendizaje por transferencia.

La figura 3.7 muestra la eficacia del agente para encontrar el nuevo objetivo de la prueba tras la convergencia inicial, lo que indica que el agente de pruebas es capaz de reutilizar adecuadamente la política aprendida en escenarios de prueba similares, es decir, en episodios con objetivos de rendimiento diferente y aún así cumplir de manera eficiente con el objetivo de la prueba.

Capítulo 4

Evaluación

4.1. Diseño de la evaluación

La evaluación diseñada para este proyecto estableció en primera instancia la experimentación de las diferentes configuraciones definidas en la tabla 4.1, para analizar la eficiencia del agente de pruebas de carga y así obtener la configuración con mejor convergencia en la politica de aprendizaje y alcanzar el objetivo de rendimiento propuesto, el cuál establece que el tiempo de respuesta del sistema sujeto a pruebas (SUT) supere los 1500 milisegundos o la tasa de error de las transacciones sea mayor al 10 %.

Tabla 4.1: Listado de experimentos.

Experimento	**Configuración**
1	$\alpha = 0.5$, $\gamma = 0.1$ y $\varepsilon =$ decaying
2	$\alpha = 0.5$, $\gamma = 0.3$ y $\varepsilon =$ decaying
3	$\alpha = 0.5$, $\gamma = 0.5$ y $\varepsilon =$ decaying
4	$\alpha = 0.5$, $\gamma = 0.7$ y $\varepsilon =$ decaying
5	$\alpha = 0.5$, $\gamma = 0.9$ y $\varepsilon =$ decaying
6	$\alpha = 0.1$, $\gamma = 0.5$ y $\varepsilon =$ decaying
7	$\alpha = 0.3$, $\gamma = 0.5$ y $\varepsilon =$ decaying
8	$\alpha = 0.7$, $\gamma = 0.5$ y $\varepsilon =$ decaying
9	$\alpha = 0.9$, $\gamma = 0.5$ y $\varepsilon =$ decaying
10	$\alpha = 0.5$, $\gamma = 0.5$ y $\varepsilon = 0.2$
11	$\alpha = 0.5$, $\gamma = 0.5$ y $\varepsilon = 0.8$

Posteriormente, se realizó una evaluación de los resultados obtenidos de la configuración que mejor eficiencia presentó en el estudio de experimentación, en comparación con una nueva ejecución del agente de pruebas con los mismos parámetros, pero realizando cambios a los datos inicialmente definidos para las pruebas, en este caso se buscó generar información de manera aleatoria, y así, analizar si los resultados obtenidos eran coherentes con la

convergencia de la configuración inicial.

Por último se análizaron los resultados del agente, respecto a la ajecución de una prueba de carga estándar (Base line), generando una carga de trabajo inicial con todas las transacciones definidas en el plan de pruebas con el mismo número de usuarios virtuales para cada transacción y posteriormente aumentando la carga (usuarios virtuales) en un 33 % hasta alcanzar el objetivo.

4.2. Resultados de la evaluación

Para los resultados de la evaluación de la fase de aprendizaje inicial, se compararon las diferentes configuraciones para analizar la eficiencia del agente. Para las configuraciones 1, 4, 6, 8 y 11 (tabla 4.1), se observan formas en picos arriba y abajo finalizando los episodios propuestos, lo que nos indica que estas configuraciones no convergen en la política óptima o requieren más episodios para obtener el resultado. Para las configuraciones 2, 5, 7 y 9 se puede concluir que, al final de los 40 episodios empieza a estabilizar la carga de trabajo, lo que implica que necesita algunos episodios extras. La configuración 10 hace que el agente confíe mas en la experiencia almacenada, en lugar de explorar nuevas acciones, lo que relentiza la convergencia en un entorno desconocido que requiere más exploración. Por último, la configuración 3 converge después de los 32 episodios, como se puede ver en la figura 4.2.

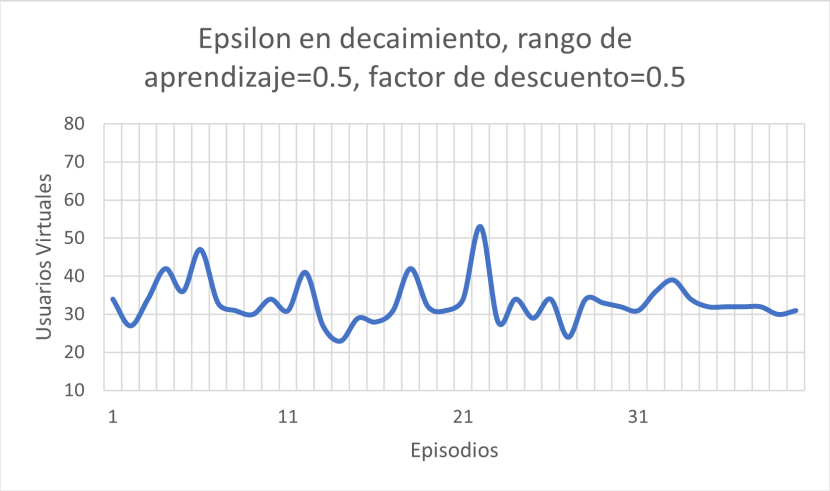


Figura 4.1: Eficacia del agente para la configuración 3.

Los diferentes experimentos muestran como los hiperparámetros influyen en el aprendizaje del agente. En conclusión, el agente no converge con una tasa de aprendizaje baja,

además, con tasas de descuento mas altas y bajas la convergencia del agente es mas lenta. Las tablas Q para cada configuración se pueden evidenciar en la sección 6.3 y las graficas de los resultados en la sección 6.4 de los anexos.

Teniendo en cuenta los resultados de eficiencia del agente, se optó por utilizar la configuración 3 (es decir, $\alpha = 0.5$, $\gamma = 0.5$ y $\varepsilon =$ decaying) para realizar la fase de aprendizaje por transferencia descrita en la sección 3.3.3 y la evaluación con respecto a la prueba de carga estandar. Esta configuración arrojó mejores resultados en la convergencia de la política, estableciendo el valor de exploración y explotación decreciente, lo que genera una disminución de ε a lo largo de los episodios de aprendizaje, buscando tener más exploración en los primeros episodios y explotar la política aprendida en los episodios finales, además el mismo valor para la tasa de aprendizaje y el factor de descuento.

Para evaluar la configuración con mejor eficiencia, se realizaron nuevas ejecuciones del agente de pruebas, cambiando la información de cada transacción del plan de pruebas en JMeter, esto con el fin de analizar los resultados del agente con diferentes datos de prueba y validar los resultados obtenidos en la experimentación. Durante los mismos 40 episodios y bajo el mismo objetivo de rendimiento, donde el tiempo de respuesta del sistema sujeto a pruebas (SUT) supere los 1500 milisegundos o la tasa de error de las transacciones sea mayor al 10 %, se evidenció que el agente converge despues de los 34 episodios para la misma configuración y datos desconocidos, esto da validez a los resultados obtenidos en la experimentación.

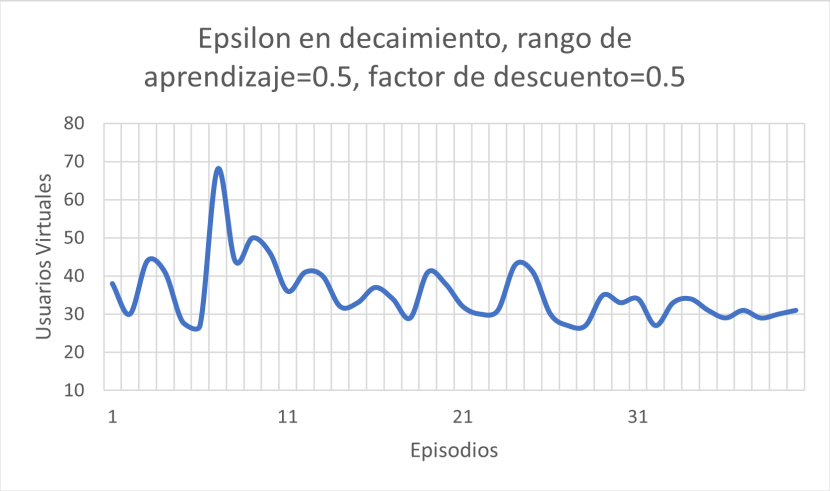


Figura 4.2: Evaluación de la configuración del agente con datos desconocidos.

Para evaluar los resultados de la configuración del agente, se realizó una prueba de carga

estándar, aplicando una carga de trabajo inicial a todas las transacciones definidas en el plan de pruebas y posteriormente aumentando la carga (usuarios virtuales) en un 33 % hasta alcanzar el objetivo y durante el mismo número de episodios, como se puede observar en la figura 4.3.

Figura 4.3: Resultados prueba de carga estándar.

Tomando los últimos 10 episodios de la prueba realizada con el agente y la prueba estándar, en el primero la carga de trabajo promedio generada para alcanzar el objetivo de la prueba fue de 36 usuarios virtuales, mientras que en la prueba de carga estándar fue de 48 usuarios virtuales, lo que implica una reducción del 25 % de la carga generada para cumplir con el objetivo de rendimiento. Estos resultados permitieron concluir que al agente de pruebas es capaz de generar cargas mas precisas y ajustadas a los objetivos de rendimiento de la prueba, lo que se traduce en métricas más precisas sobre el estado del sistema sujeto a pruebas y las necesidades de rendimiento del negocio. Además, el agente de prueba puede aprender la forma de generar cargas de manera efectiva y explotarla en otros escenarios de prueba. Por otro lado, respecto a herramientas para pruebas de pago, el agente de pruebas puede ayudar en el ahorro del costo de las pruebas de carga, ya que algunas de las herramientas realizan el cobro por la cantidad de cargas o usuarios generados.

Finalmente, podemos encontrar que los requisitos de rendimiento no siempre están definidos o son claros respecto al negocio, en muchas ocasiones no hay claridad de cuantos usuarios finales van a usar el sistema sujeto a pruebas, lo que implica generar pruebas de carga sin una estrategia definida, aquí el agente de pruebas puede ser una sólida ayuda para identificar los usuarios que el sistema sujeto a pruebas puede soportar bajo un tiempo de respuesta o tasa de error específico.

Capítulo 5

Conclusiones

5.1. Conclusiones

En este proyecto se empleó un agente inteligente para pruebas de carga llamado RELOAD, para evaluar el rendimiento de una aplicación web en desarrollo. Esto implicó realizar un análisis de las técnicas de aprendizaje por refuerzo y Q-learning, conocer el funcionamiento y configuración del agente de pruebas para soportar el sistema sujeto a pruebas definido, para finalmente realizar una evaluación experimental de los resultados de cada ejecución de pruebas y así determinar la eficiencia del agente para realizar pruebas de carga.

El proceso de análisis y configuración del agente de pruebas de carga inteligente, puede presentar desafíos significativos debido a la falta de documentación, la complejidad inherente del código fuente y la limitada parametrización del mismo, puede dificultar la comprensión y la implementación efectiva del agente, lo que puede resultar en un proceso denso y complicado para los ingenieros de pruebas y el equipo de desarrollo. Sin embargo, superar estos obstáculos y lograr una configuración exitosa del agente puede generar numerosos beneficios en el proceso de pruebas de rendimiento. Al implementar adecuadamente el agente, se puede mejorar la eficiencia de las pruebas de carga y obtener una comprensión más profunda del rendimiento del sistema sujeto a pruebas en entornos diversos. A pesar de los desafíos iniciales, el esfuerzo invertido en analizar y configurar el agente puede resultar en una mayor calidad del software y una mejora significativa en la eficiencia del proceso de pruebas de carga.

El uso del agente de pruebas de carga inteligente, impulsado por aprendizaje por refuerzo, ofrece una serie de ventajas significativas en el proceso de pruebas de rendimiento. Este enfoque permite una generación eficiente de cargas de trabajo de prueba efectivas, reduciendo la dependencia de modelos complejos y del código fuente. Además, la capacidad de adaptación del agente a cambios en el entorno de ejecución y su habilidad para reutilizar políticas aprendidas en diferentes escenarios de prueba proporcionan una mayor flexibilidad y eficiencia en el proceso de pruebas.

En última instancia, el agente de pruebas de carga tiene el potencial de reducir el esfuerzo y los costos asociados con las pruebas de rendimiento, lo que lo convierte en una herramienta

valiosa en el ciclo de vida de desarrollo de software, cómo en las pruebas de regresión, permitiendo asegurar los requisitos de rendimiento de una manera eficiente, teniendo en cuenta que es importante considerar que no todas las transacciones tienen el mismo efecto en el rendimiento, es decir, ajustar la carga de trabajo de manera óptima es crucial para la eficiencia de las pruebas de carga.

5.2. Trabajos futuros

Para futuros trabajos, se propone explorar la integración del agente de pruebas de carga inteligente, en los procesos de Integración Continua (CI) y Despliegue Continuo (CD). Esta integración permitiría automatizar aún más el proceso de pruebas de rendimiento dentro del ciclo de desarrollo de software, lo que conduciría a una mejora significativa en la calidad y el rendimiento de las aplicaciones web en desarrollo.

En primer lugar, se podría diseñar un flujo de trabajo automatizado que incorpore la ejecución de pruebas de carga inteligentes como parte del proceso de CI/CD. Esto garantizaría que cada vez que se realice un cambio en el código fuente, se ejecuten automáticamente pruebas de rendimiento para evaluar el impacto en el sistema. Esto proporcionaría retroalimentación inmediata a los desarrolladores sobre el rendimiento de sus cambios, lo que les permitiría identificar y abordar posibles problemas de rendimiento de manera proactiva.

Además, se podría desarrollar una infraestructura de pruebas automatizada que permita la escalabilidad y la distribución de las pruebas de carga inteligentes en entornos de CI/CD. Esto garantizaría que las pruebas puedan ejecutarse de manera eficiente en diferentes entornos y configuraciones, lo que facilitaría la detección temprana de problemas de rendimiento en diferentes etapas del ciclo de desarrollo.

Otro aspecto importante a considerar es la integración de métricas de rendimiento automatizadas en los procesos de CI/CD. Esto incluiría la recopilación y el análisis automatizado de datos de rendimiento durante las pruebas de carga inteligentes, así como la generación de informes detallados sobre el rendimiento del sistema. Estos informes podrían utilizarse para evaluar la estabilidad de la aplicación en desarrollo y proporcionar información valiosa para la toma de decisiones sobre la implementación.

En resumen, la aplicación del agente de pruebas de carga inteligente en procesos de CI/CD tiene el potencial de mejorar significativamente la calidad y el rendimiento del software en desarrollo al automatizar las pruebas de rendimiento y proporcionar retroalimentación rápida al equipo de desarrollo. Esto podría conducir a una mayor eficiencia en el desarrollo

de software y una mejor experiencia del usuario final.

5.3. Lecciones aprendidas

Durante el desarrollo del proyecto de pruebas de carga utilizando un agente inteligente, se han identificado varias lecciones clave que podrían servir como guía para futuros proyectos similares:

1. **Importancia de la capacitación en RL y Q-Learning:** Previo al proceso de análisis y configuración del agente de pruebas de carga inteligente, se hizo evidente la importancia de contar con un entendimiento sólido de los conceptos de aprendizaje por refuerzo y Q-Learning, que son la base del agente. La capacitación adecuada en estos conceptos es esencial para comprender cómo funciona el agente y cómo configurarlo correctamente para lograr resultados óptimos en las pruebas de carga.

2. **Importancia de la documentación:** La documentación actuál del agente de pruebas de carga inteligente puede dificultar el proceso de análisis y configuración. Sería importante contar con una documentación más detallada que describa el funcionamiento del agente, sus parámetros y su integración con el entorno de pruebas.

3. **Complejidad en el código fuente y parametrización:** El análisis y configuración del agente pueden verse obstaculizados por la complejidad del código fuente. Es fundamental comprender en profundidad el funcionamiento interno del agente y estar preparado para abordar posibles desafíos técnicos durante el proceso de configuración.

4. **Beneficios de la automatización:** A pesar de los desafíos mencionados anteriormente, la aplicación del agente de pruebas de carga inteligente ofrece numerosos beneficios, como la capacidad de generar cargas de trabajo eficientes y reutilizar políticas de prueba aprendidas en diferentes escenarios. La automatización de las pruebas de carga puede mejorar significativamente la calidad y el rendimiento del software en desarrollo.

5. **Contribución a la calidad del software:** El uso del agente de pruebas de carga inteligente puede contribuir a mejorar la calidad del software al asegurar los requisitos de rendimiento de manera eficiente. Además de su aplicación en pruebas de carga de regresión, puede ser útil garantizando que los cambios en el software no afecten negativamente su rendimiento.

El proceso de análisis y configuración del agente de pruebas de carga inteligente resaltó la importancia de conocer los fundamentos de aprendizaje por refuerzo (RL) y Q-Learning. Una

comprensión sólida de estos conceptos es esencial para maximizar la comprensión del funcionamiento del agente de pruebas de carga. Aunque la documentación actual del agente puede ser limitada, superar la complejidad de comprensión del código fuente y la parametrización del mismo, conlleva a la automatización de las pruebas de carga, que ofrece una variedad de beneficios, desde la generación eficiente de cargas de trabajo hasta la reutilización de políticas de prueba en diferentes escenarios. En última instancia, la aplicación del agente no solo mejora la calidad del software, sino también la eficiencia del proceso de pruebas, lo que representa una valiosa contribución al ciclo de vida de desarrollo de software. Estas lecciones aprendidas pueden servir como base para futuros proyectos y contribuir a las prácticas de pruebas de carga en el desarrollo de software.

Bibliografía

Abdul, M., Abdul, M., and Mubashir, K. (2020). Performance testing framework for software mobile applications. *International Journal of Innovative Research in Science, Engineering and Technology (IJIRSET)*, 9.

Aberle, M. (2022). *Adaptive Learning in an Interactive Teaching Platform*. PhD thesis.

Boussakssou, M., Hssina, B., and Erittali, M. (2020). Towards an adaptive e-learning system based on q-learning algorithm. *Procedia Computer Science*, 170:1198–1203. The 11th International Conference on Ambient Systems, Networks and Technologies (ANT) / The 3rd International Conference on Emerging Data and Industry 4.0 (EDI40) / Affiliated Workshops.

Chen, H. and Hossain, M. (2022). Application of machine learning on software quality assurance and testing: A chronological survey. In Gupta, B., Bandi, A., and Hossain, M., editors, *Proceedings of 37th International Conference on Computers and Their Applications*, volume 82 of *EPiC Series in Computing*, pages 42–52. EasyChair.

Fan, C., Yao, L., Zhang, J., Zhen, Z., and Wu, X. (2023). Advanced reinforcement learning and its connections with brain neuroscience. *Research*, 6.

Flórez Tunaroza, D. J. S. (2016). Procedimiento para la aplicación de pruebas de carga y rendimiento mediante una herramienta tecnológica en aplicativo web para el ciadti. Publicado.

Gil-Vera, V. D. and Seguro-Gallego, C. (2022). Machine learning aplicado al análisis del rendimiento de desarrollos de software. *Revista Politécnica*, 18(35):128–139.

Helali Moghadam, M. (2020). Machine learning-assisted performance assurance.

Hernández, P., Marrero Alemán, M., Paz Hernández, R., Bordón Pérez, P., Suárez, L., and Benítez-Vega, A. (2019). *Adaptive Learning Using Interactive Training Material*, pages 162–184.

Irrazábal, E. and Mascheroni, M. (2022). *Fundamentos de las pruebas continuas de software*. Editorial de la Universidad Nacional del Nordeste EUDENE.

Jayasiriwardene, S. and Meedeniya, D. (2023). An adaptive and interactive learning toolkit (ilearn). *Software Impacts*, 15:100471.

Jiang, Z. M., Hassan, A. E., Hamann, G., and Flora, P. (2009). Automated performance analysis of load tests. In *2009 IEEE International Conference on Software Maintenance*, pages 125–134.

Jiang, Z. M. J. (2015). Load testing large-scale software systems. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 2, pages 955–956.

Lascorz, L. and Alcalá, J. (2018). Aprendizaje por Refuerzo. Elementos básicos y algoritmos. Universidad de Zaragoza.

Lawanna, A. (2012). The theory of software testing. *Intelligent Transportation Systems Journal*, 16:35–40.

Li, Z., Shi, L., Yang, L., and Shang, Z. (2020). *An Adaptive Learning Rate Q-Learning Algorithm Based on Kalman Filter Inspired by Pigeon Pecking-Color Learning*, pages 693–706.

Lizcano, A., García, J., and Pérez Holguín, R. (2009). Tendencias en el desarrollo de agentes inteligentes aplicados en la construcción de un aula inteligente. *Revista de Investigaciones UNAD*, 8:85.

Mikalef, P., Fjørtoft, S., and Torvatn, H. (2019). Developing an artificial intelligence capability: A theoretical framework for business value.

Moghadam, M. H., Hamidi, G., Borg, M., Saadatmand, M., Bohlin, M., Lisper, B., and Potena, P. (2021). Performance testing using a smart reinforcement learning-driven test agent. In *2021 IEEE Congress on Evolutionary Computation (CEC)*, pages 2385–2394.

Navaei, M. and Tabrizi, N. (2022). Machine learning in software development life cycle: A comprehensive review. pages 344–354.

Ortiz, C. and Duque, M. (2022). Software para la validación de pruebas de carga y estrés en sistemas de información web. Universidad Tecnologica de Santander.

Peñafiel, M. (2022). Definición y automatización de pruebas de carga y escalabilidad para una aplicación web colaborativa. Universidad de Valladolid. Escuela de Ingeniería Informática de ValladolidAutoridad UVA.

Ponce, J., Torres, A., Aguilera, F., Silva Sprock, A., Flor, E., Casali, A., Scheihing, E., Tupac, Y., Torres, D., Ornelas, F., Hernández-Aguilar, J. A., D., C., Vakhnia, N., and Pedreño, O. (2014). *Inteligencia Artificial*.

Poole, D. and Mackworth, A. (2010). *Artificial Intelligence: Foundations of Computational Agents*.

Rouhiainen, L. P. (2018). *Inteligencia artificial : 101 cosas que debes saber hoy sobre nuestro futuro*. Alienta Editorial., 2st edition.

Rudkowskyj Hernanz, S. (2019). Aprendizaje por refuerzo en sistemas robóticos. No Publicado.

Sánchez Peño, J. M. (2015). Pruebas de software. fundamentos y técnicas. Universidad Politécnica de Madrid.

Singh, Y. (2022). Implementation of machine learning techniques to overcome the challenges of performance testing.

Tappler, M., Aichernig, B. K., Bacci, G., Eichlseder, M., and Larsen, K. G. (2019). L*-based learning of markov decision processes (extended version).

Thongprasit, J. and Wannapiroon, P. (2022). Framework of artificial intelligence learning platform for education. *International Education Studies*, 15:76.

Whiting, E. and Datta, S. (2021). Performance testing and agile software development: A systematic review.

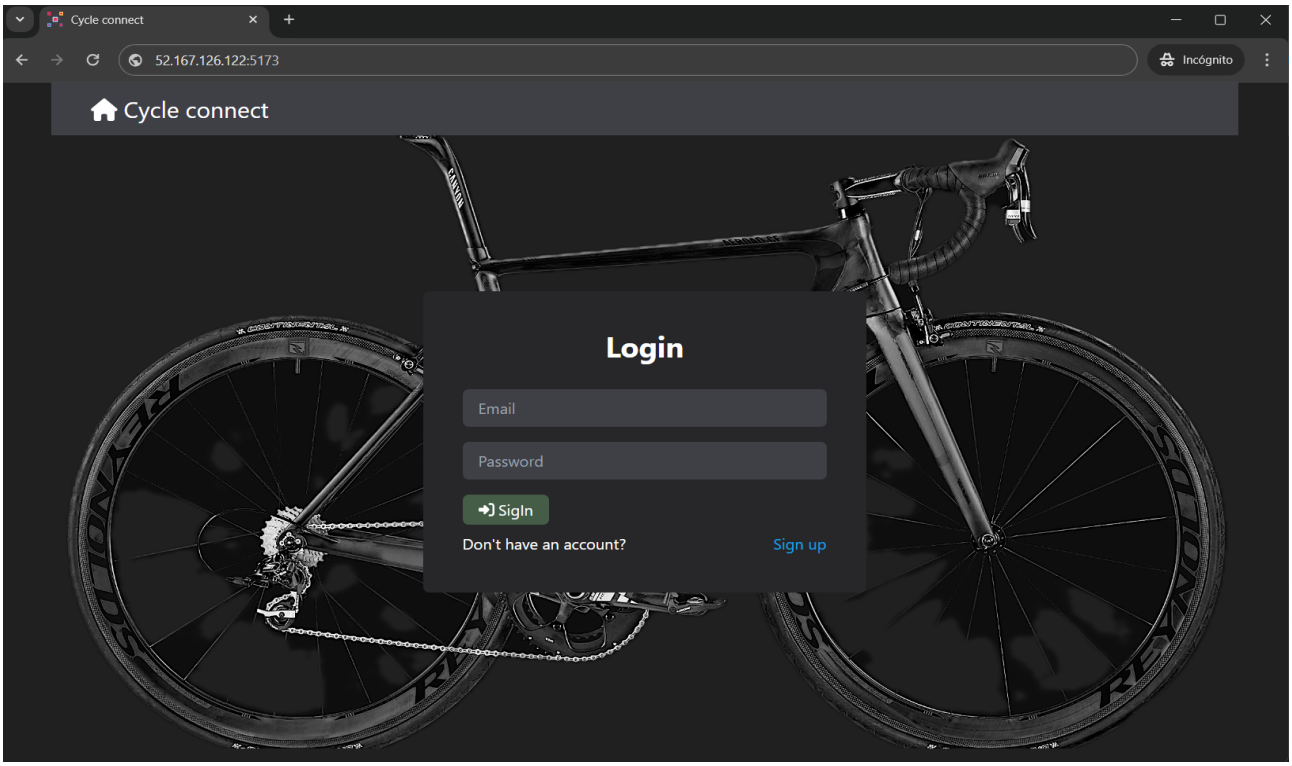
Zhang, L., Tang, L., Zhang, S., Wang, Z., Shen, X., and Zhang, Z. (2021). A self-adaptive reinforcement-exploration q-learning algorithm. *Symmetry*, 13:1057.

Capítulo 6

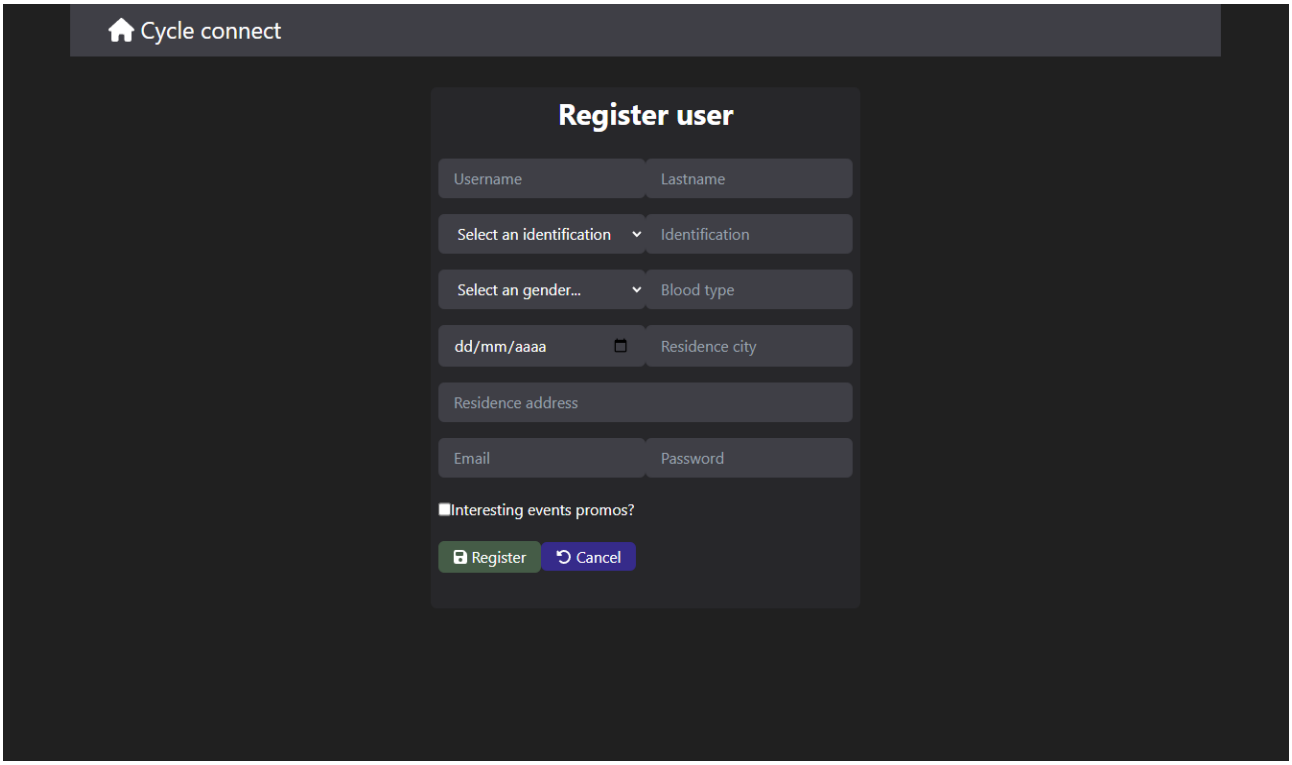
Anexos

6.1. Sistema Sujeto a Pruebas (SUT)

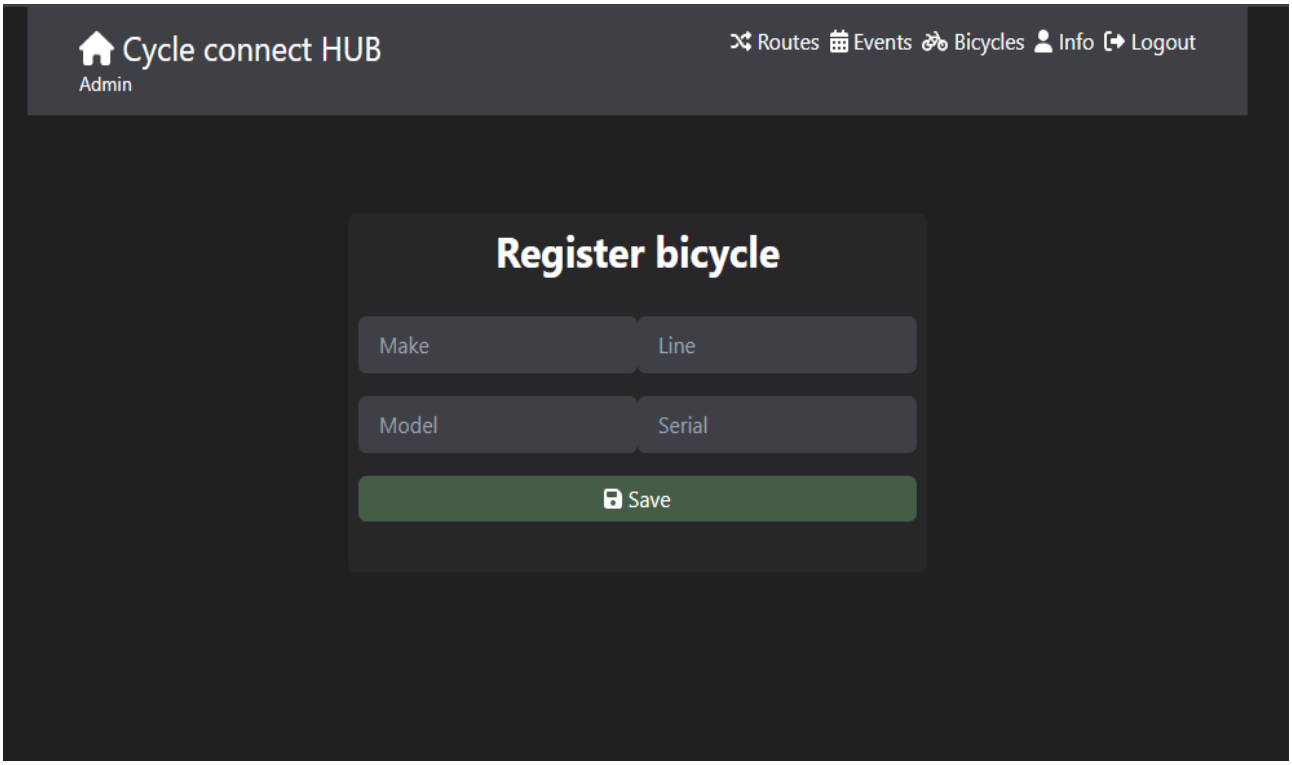
6.1.1. Inicio de sesión



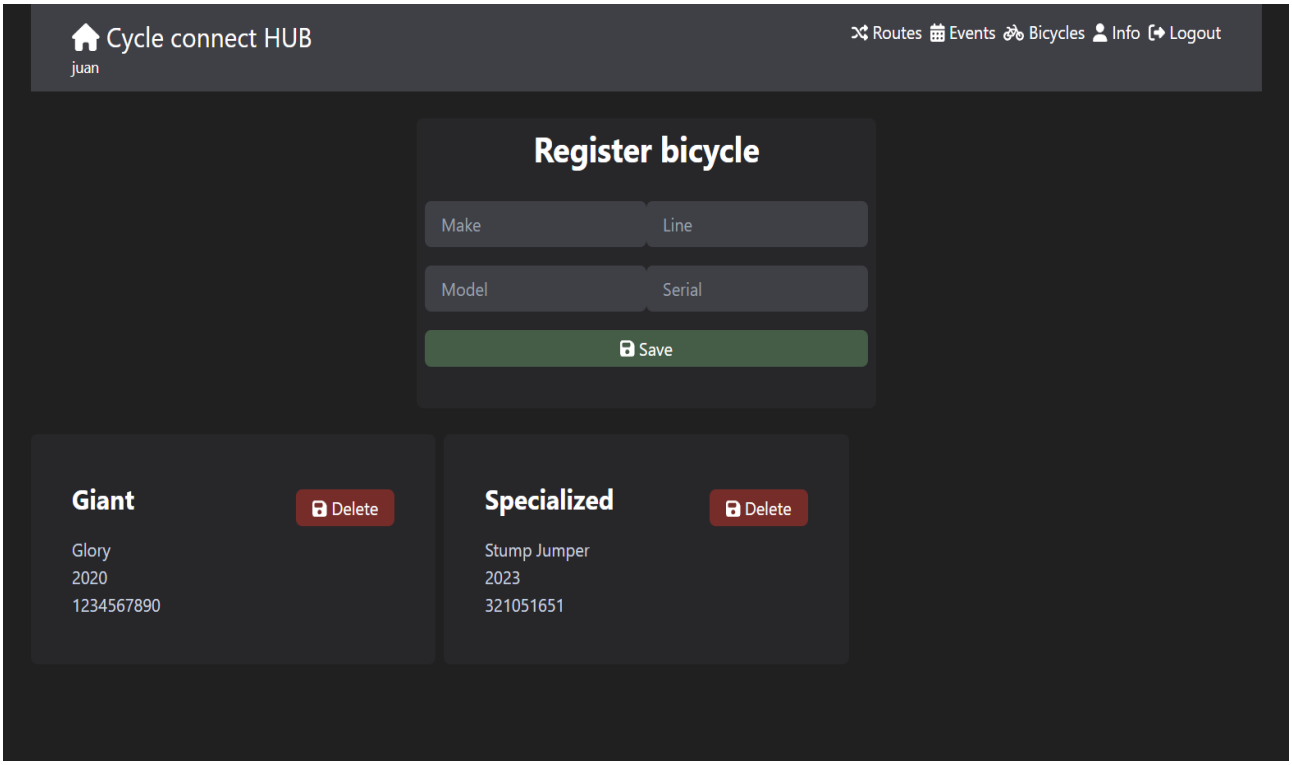
6.1.2. Registro de usuario



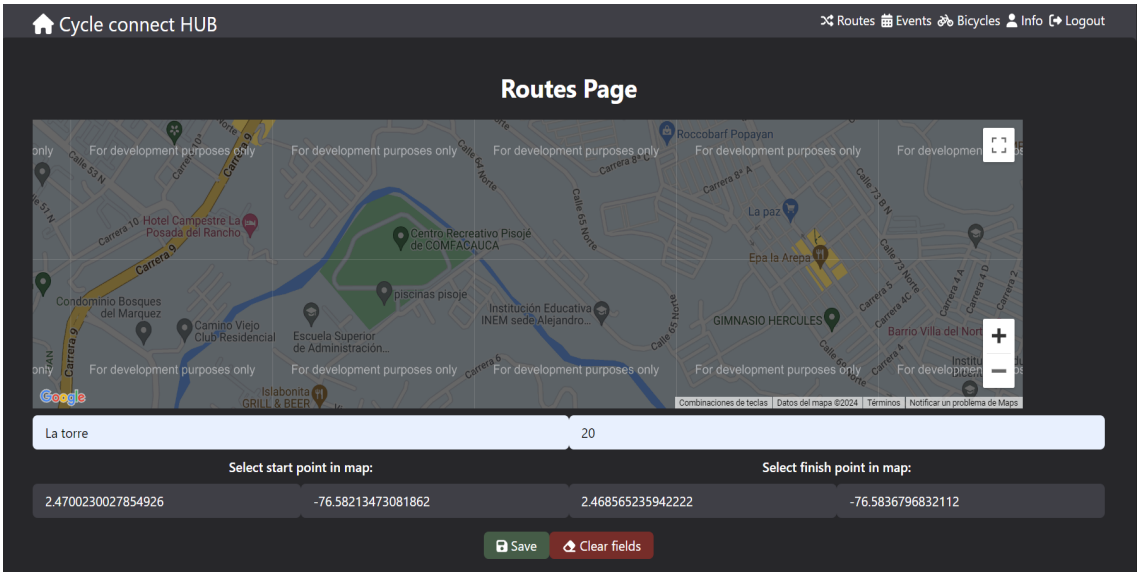
6.1.3. Registro de bicicletas

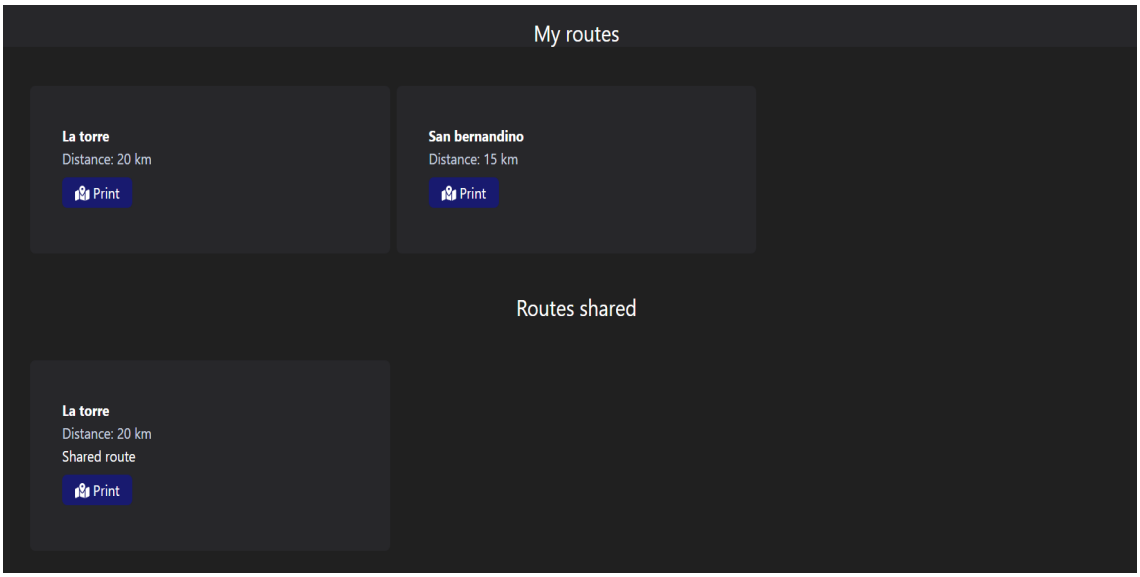


6.1.4. Obtener bicicletas

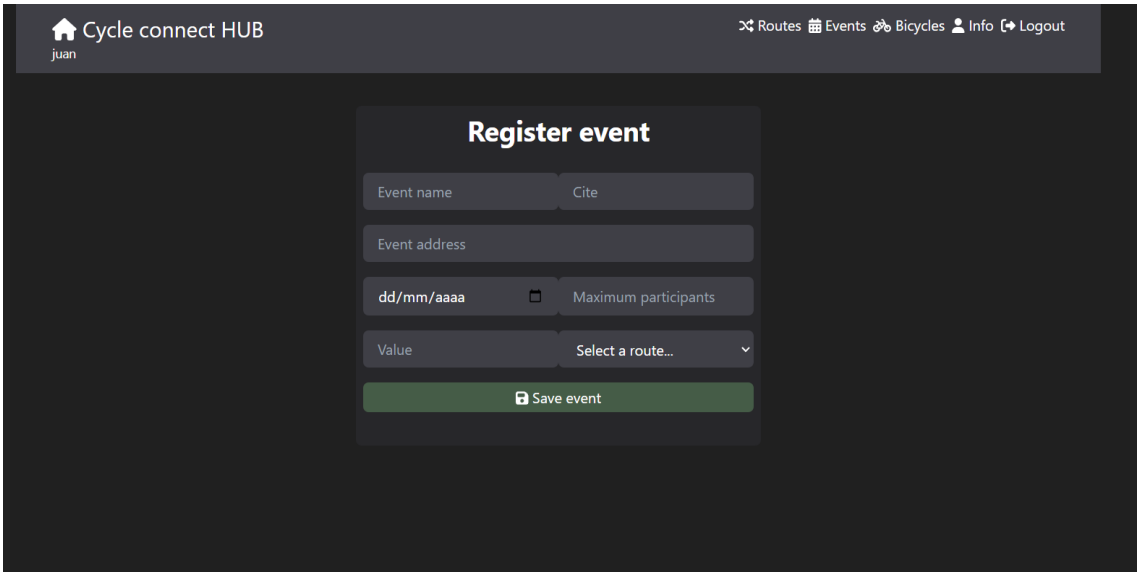


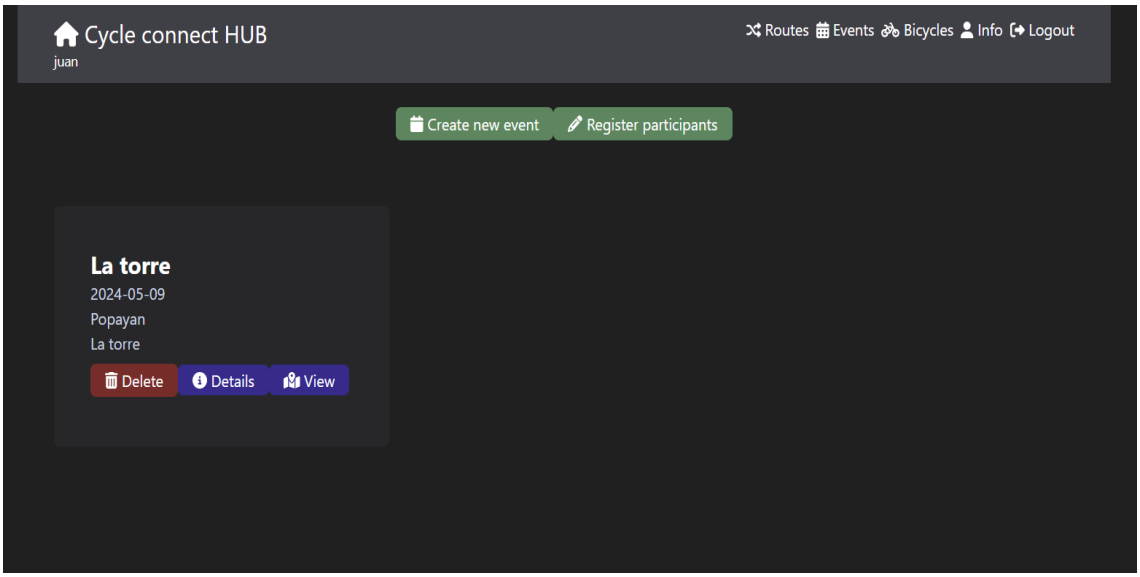
6.1.5. Registro de rutas



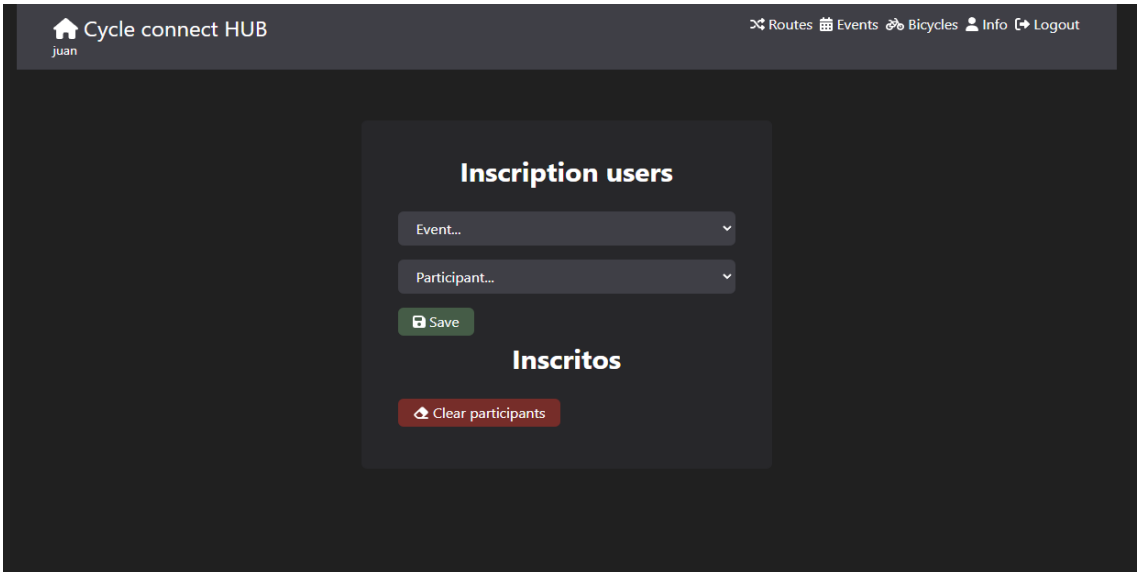


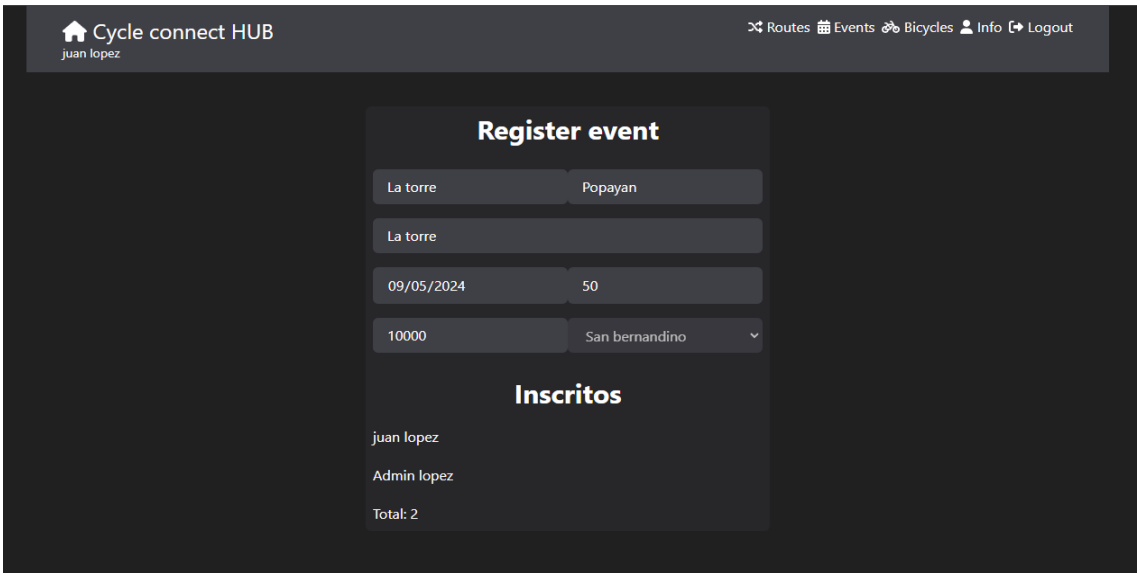
6.1.6. Registro de eventos





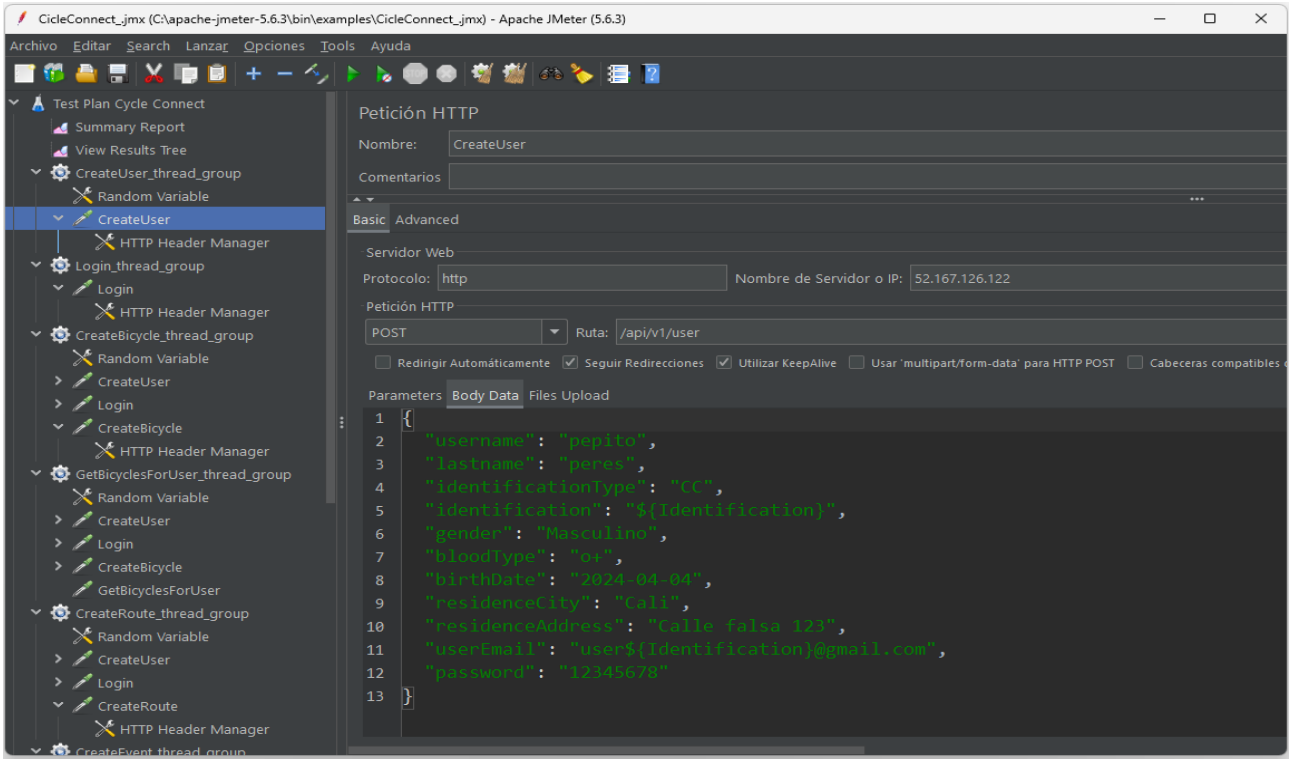
6.1.7. Registro de participantes



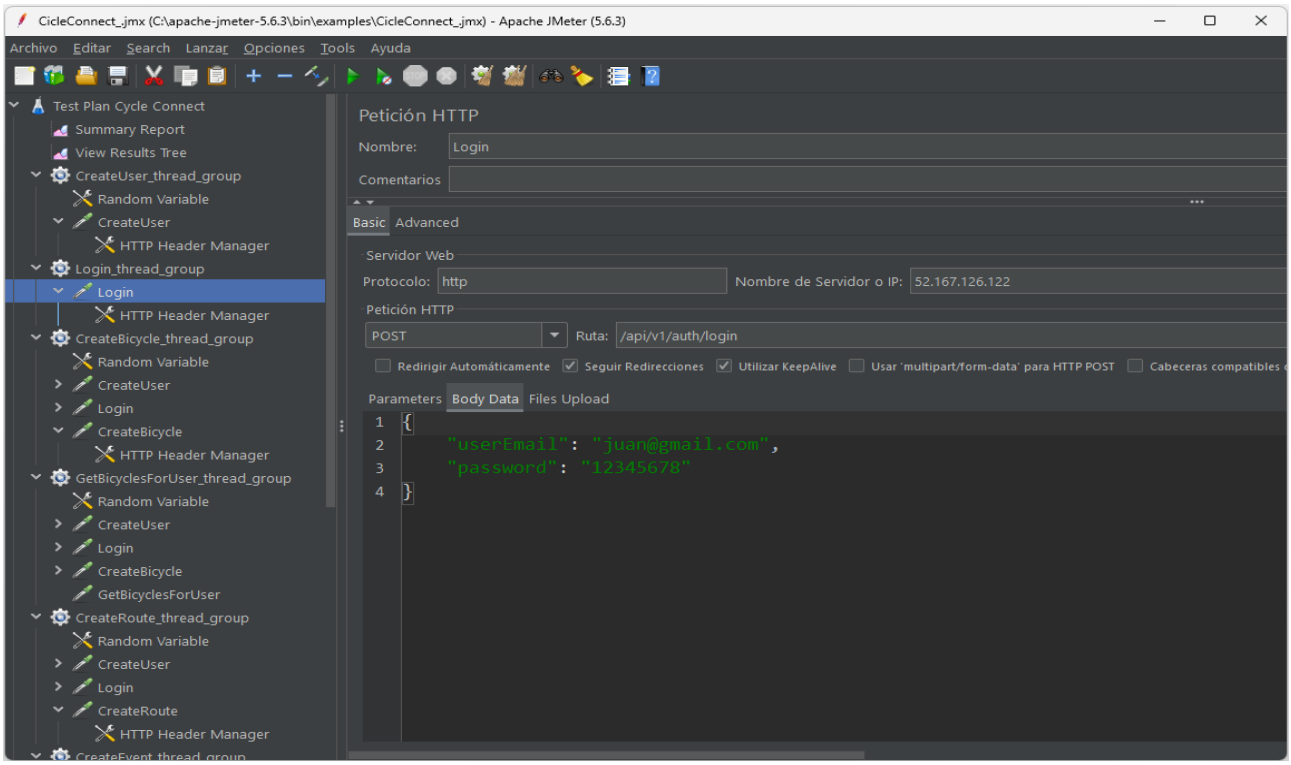


6.2. Plan de pruebas

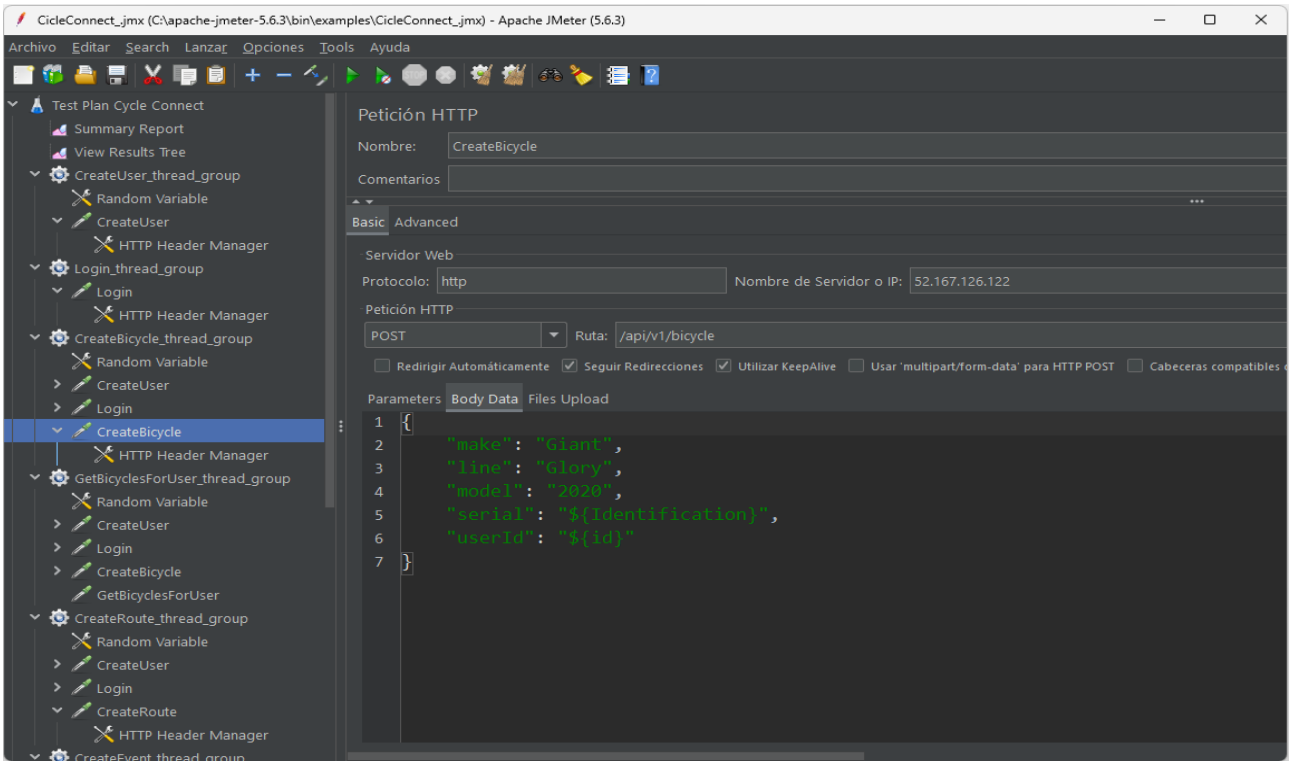
6.2.1. Transacción registro de usuario



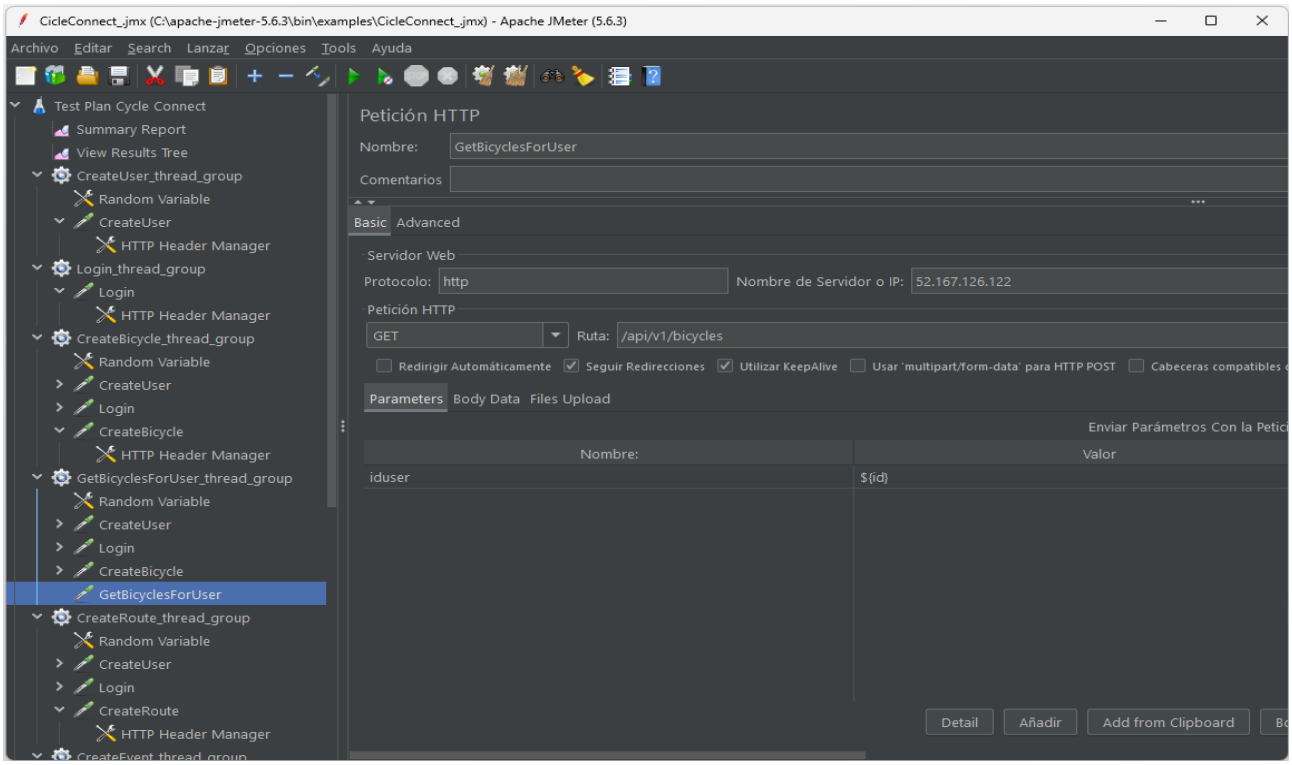
6.2.2. Transacción inicio de sesión



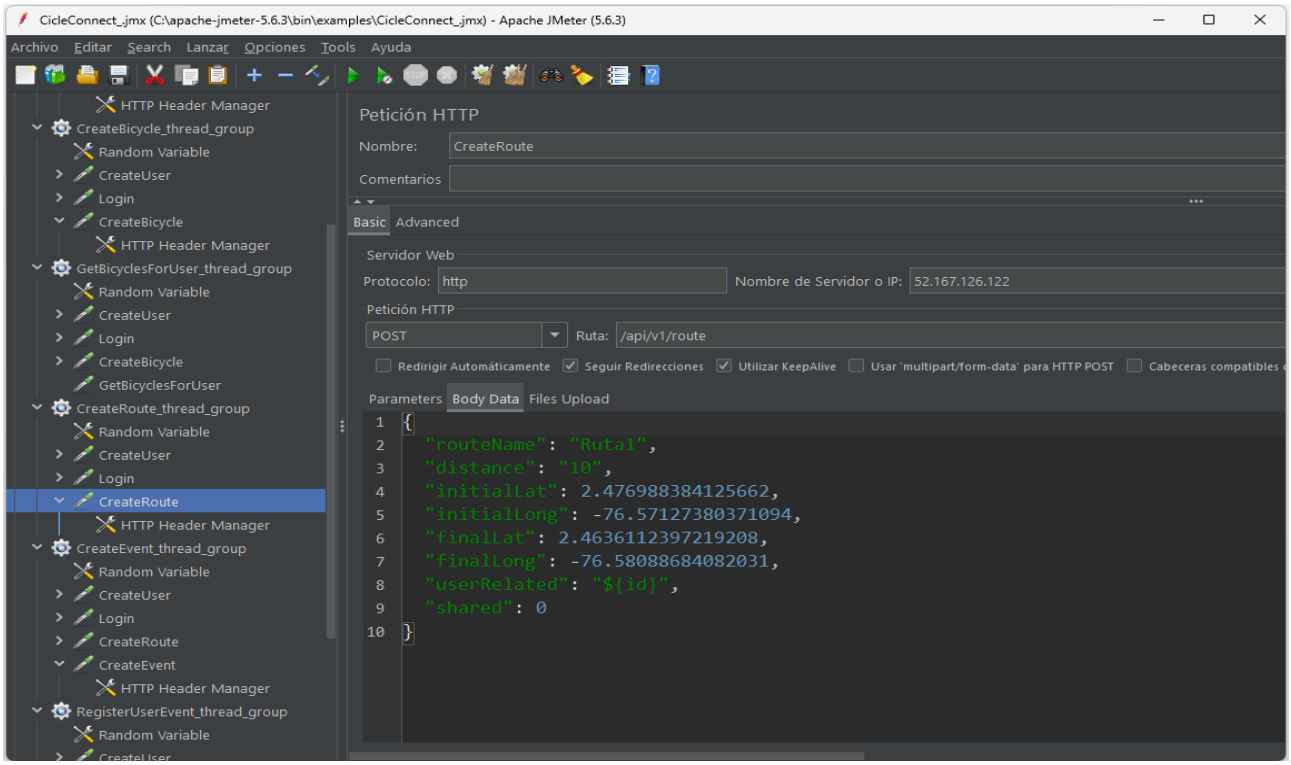
6.2.3. Transacción registro de bicicletas



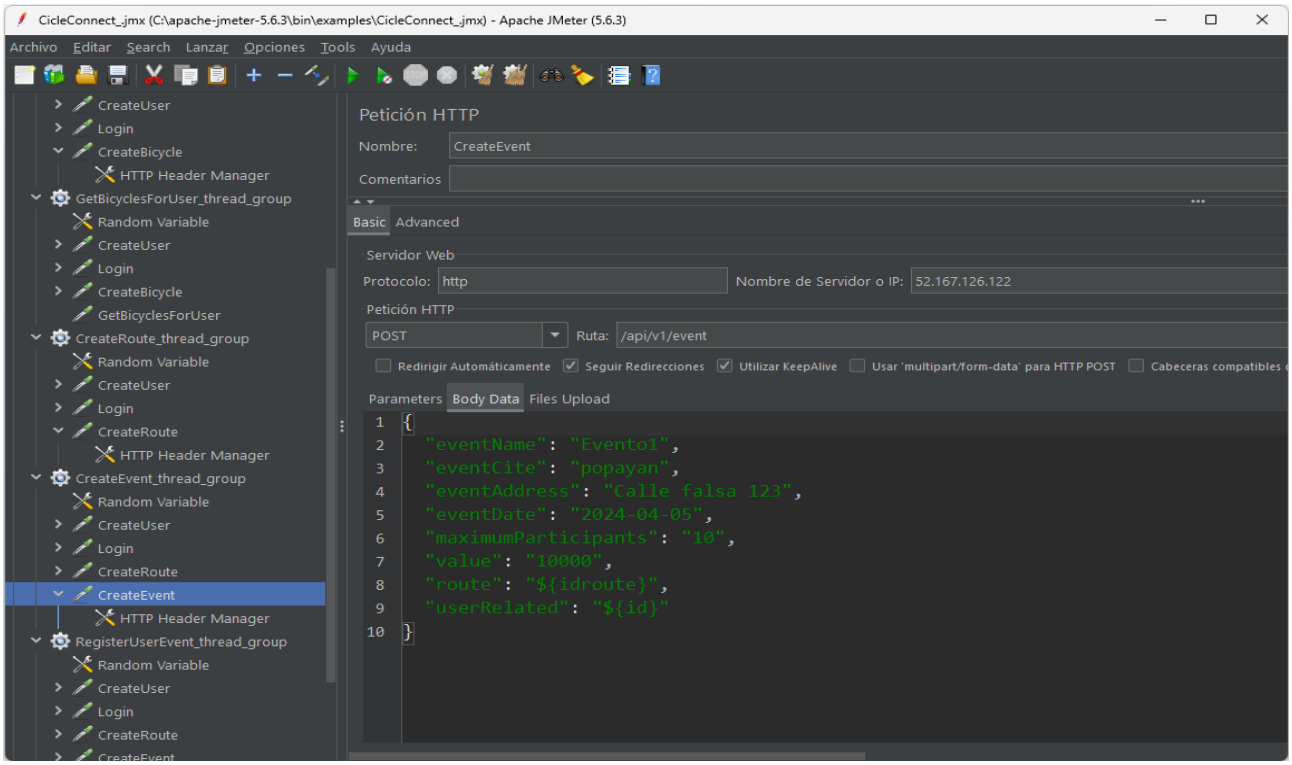
6.2.4. Transacción obtener bicicletas



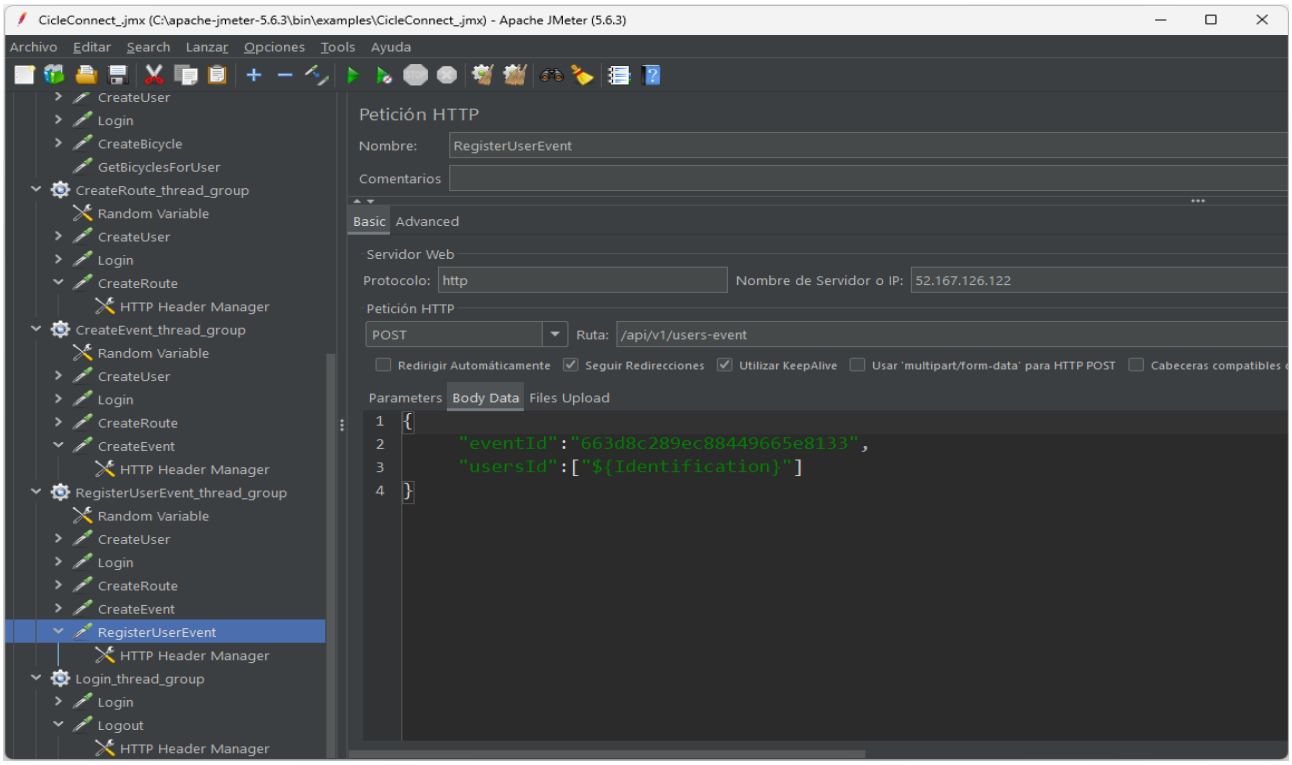
6.2.5. Transacción registro de rutas



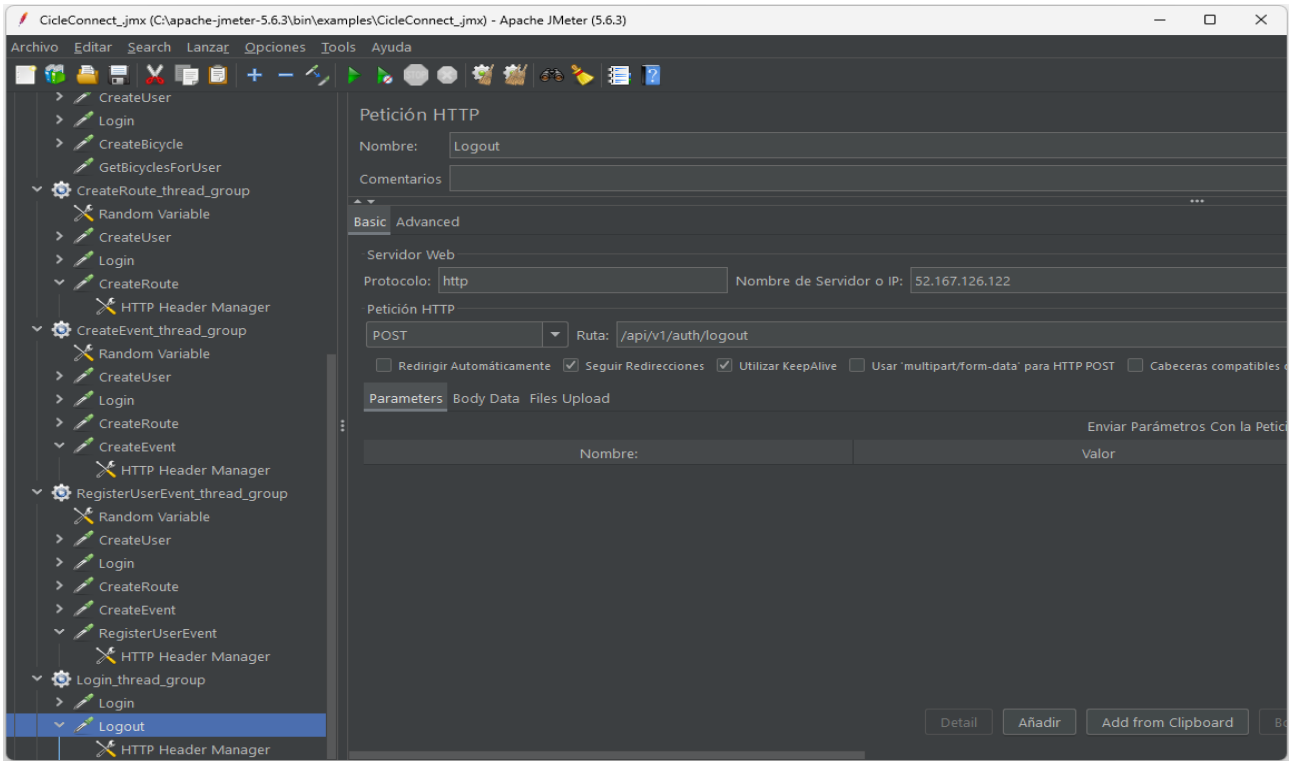
6.2.6. Transacción registro de eventos



6.2.7. Transacción registro de participantes



6.2.8. Transacción cerrar sesión



6.3. Tablas Q-Learning

6.3.1. Agente con $\alpha = 0.5$, $\gamma = 0.1$ y $\varepsilon =$ decaying

Tiempo max. respuesta	Tasa max. error	Retraso episodio	Rango aprendizaje	Factor descuento
1500	0.1	2	0.5	0.5
Episodio	**Carga de trabajo**	**Paso episodio**	**Tiempo respuesta**	**Tasa error**
1	37	14	1719.3235294117646	0.0
2	40	18	1703.0	0.0
3	37	18	1600.49	0.0
4	42	18	1563.5348837209303	0.0
5	34	14	2074.6039603960394	0.0
6	60	23	2339.700787401575	0.0
7	42	20	2197.1559633027523	0.0
8	31	13	1718.1707317073171	0.0
9	32	14	1749.4302325581396	0.0
10	49	23	2022.2325581395348	0.0
11	44	20	2146.2169811320755	0.0
12	31	13	1708.0365853658536	0.0
13	47	18	1988.9583333333333	0.0
14	44	19	1578.8482142857142	0.0
15	39	17	2187.3809523809523	0.0
16	37	17	1589.2293577981652	0.0
17	24	6	1576.6666666666667	0.0
18	25	7	2294.68115942029	0.0
19	20	2	1744.5	0.0
20	41	19	1553.8454545454545	0.0
21	29	10	1736.9634146341464	0.0
22	28	9	1643.9638554216867	0.0
23	35	13	1862.7191011235955	0.0
24	23	5	1545.111111111111	0.0
25	41	19	1538.4601769911505	0.0
26	38	16	1514.9655172413793	0.0
27	28	9	1698.8101265822784	0.0
28	24	5	1589.623188405797	0.0

Continúa en la siguiente página

Tabla 6.1 – continúa desde la página anterior

Episode	TotalWorkload	EpisodeStep	ResponseTime	ErrorRate
29	33	13	1501.4329896907216	0.0
30	38	16	2035.5567010309278	0.0
31	32	10	1513.388888888889	0.0
32	33	11	1554.0	0.0
33	33	11	1532.9302325581396	0.0
34	31	9	1964.1978021978023	0.0
35	30	10	1578.1976744186047	0.0
36	28	6	1524.3863636363637	0.0
37	30	8	1567.6777777777777	0.0
38	25	5	1692.4736842105262	0.0
39	21	3	1501.6	0.0
40	19	1	1541.8979591836735	0.0

6.3.2. Agente con $\alpha = 0.5$, $\gamma = 0.3$ y $\varepsilon =$ decaying

Tiempo max. respuesta	Tasa max. error	Retraso episodio	Rango aprendizaje	Factor descuento
1500	0.1	2	0.5	0.5
Episodio	**Carga de trabajo**	**Paso episodio**	**Tiempo respuesta**	**Tasa error**
1	45	22	2097.4214876033056	0.0
2	28	9	1689.0506329113923	0.0
3	37	17	1584.4888888888888	0.0
4	29	10	1666.625	0.0
5	49	24	1531.6587301587301	0.0
6	26	8	1542.4054054054054	0.0
7	32	12	1896.7096774193549	0.0
8	37	15	1617.367924528302	0.0
9	39	19	1807.6666666666667	0.0
10	42	19	1621.1916666666666	0.0
11	43	20	1614.2631578947369	0.0
12	40	18	1956.6057692307693	0.0

Continúa en la siguiente página

Tabla 6.2 – continúa desde la página anterior

Episode	TotalWorkload	EpisodeStep	ResponseTime	ErrorRate
13	38	16	2103.6190476190477	0.0
14	43	19	1690.8623853211009	0.0
15	41	15	1696.0084745762713	0.0
16	34	14	2019.225	0.0
17	44	19	1666.2072072072071	0.0
18	36	15	1808.1881188118812	0.0
19	30	10	1514.7333333333333	0.0
20	32	13	1854.6962025316457	0.0
21	31	10	1734.7564102564102	0.0
22	28	10	1511.6410256410256	0.0
23	30	11	1808.2771084337348	0.0
24	28	8	1538.8846153846155	0.0
25	24	6	1632.1803278688524	0.0
26	30	10	1993.962962962963	0.0
27	23	4	1549.6031746031747	0.0
28	35	14	1767.204081632653	0.0
29	30	10	1869.9295774647887	0.0
30	68	14	1894.3738317757009	0.0
31	30	11	1548.2328767123288	0.0
32	41	17	1877.4148936170213	0.0
33	29	10	1745.051282051282	0.0
34	34	13	2112.8045977011493	0.0
35	27	9	1618.5072463768115	0.0
36	29	7	1811.808988764045	0.0
37	30	8	1959.3666666666666	0.0
38	30	8	2089.6304347826085	0.0
39	30	8	1511.8333333333333	0.0
40	26	7	1619.6760563380283	0.0

6.3.3. Agente con $\alpha = 0.5$, $\gamma = 0.5$ y $\varepsilon =$ decaying

Tiempo max. respuesta	Tasa max. error	Retraso episodio	Rango aprendizaje	Factor descuento
1500	0.1	2	0.5	0.5
Episodio	**Carga de trabajo**	**Paso episodio**	**Tiempo respuesta**	**Tasa error**
1	34	14	2047.6326530612246	0.0
2	27	9	1529.972602739726	0.0
3	34	12	1958.366972477064	0.0
4	42	20	1549.6363636363637	0.0
5	36	17	1770.978021978022	0.0
6	47	21	1884.1203703703704	0.0
7	33	13	1851.8865979381444	0.0
8	31	12	1727.7674418604652	0.0
9	30	12	1515.6913580246915	0.0
10	34	16	2087.4886363636365	0.0
11	31	13	1628.6125	0.0
12	41	19	1507.7727272727273	0.0
13	27	8	1571.871794871795	0.0
14	23	5	1509.0338983050847	0.0
15	29	10	1712.2048192771085	0.0
16	28	10	1533.7972972972973	0.0
17	31	11	1941.888888888889	0.0
18	42	18	1515.188679245283	0.0
19	32	13	1934.4777777777779	0.0
20	31	11	1537.0210526315789	0.0
21	34	12	2011.6078431372548	0.0
22	53	23	1605.8938053097345	0.0
23	28	10	1527.7368421052631	0.0
24	34	14	1680.9878048780488	0.0
25	29	9	1665.8076923076924	0.0
26	34	13	1746.5463917525774	0.0
27	24	6	1621.1029411764705	0.0
28	34	12	1522.050505050505	0.0
29	33	10	1787.5408163265306	0.0
30	32	12	1988.89010989011	0.0

Continúa en la siguiente página

Tabla 6.3 – continúa desde la página anterior

Episode	TotalWorkload	EpisodeStep	ResponseTime	ErrorRate
31	31	11	1884.9878048780488	0.0
32	36	13	1710.8039215686274	0.0
33	39	16	1559.8648648648648	0.0
34	34	11	1586.1057692307693	0.0
35	32	10	1525.030612244898	0.0
36	32	10	1525.41	0.0
37	32	10	1500.4782608695652	0.0
38	32	10	1857.6224489795918	0.0
39	30	8	2017.891304347826	0.0
40	31	9	1562.1030927835052	0.0

6.3.4. Agente con $\alpha = 0.5$, $\gamma = 0.7$ y $\varepsilon =$ decaying

Tiempo max. respuesta	Tasa max. error	Retraso episodio	Rango aprendizaje	Factor descuento
1500	0.1	2	0.5	0.5
Episodio	**Carga de trabajo**	**Paso episodio**	**Tiempo respuesta**	**Tasa error**
1	35	15	1906.695652173913	0.0
2	36	16	2027.6237623762377	0.0
3	35	16	1636.032258064516	0.0
4	42	19	1676.733870967742	0.0
5	54	23	1605.9035087719299	0.0
6	43	17	1620.32	0.0
7	43	20	1541.2673267326732	0.0
8	44	18	1775.2845528455284	0.0
9	42	20	1709.1478260869565	0.0
10	30	12	1636.2875	0.0
11	36	15	1841.2429906542056	0.0
12	40	16	1607.967213114754	0.0
13	30	12	1779.892857142857	0.0
14	42	19	1538.111111111111	0.0

Continúa en la siguiente página

Tabla 6.4 – continúa desde la página anterior

Episode	TotalWorkload	EpisodeStep	ResponseTime	ErrorRate
15	42	20	1559.3131313131314	0.0
16	35	15	1588.164705882353	0.0
17	30	10	1618.4065934065934	0.0
18	29	10	2071.89156626506	0.0
19	31	11	1612.5421686746988	0.0
20	24	6	1534.063492063492	0.0
21	41	17	1605.188679245283	0.0
22	41	19	1514.1650485436894	0.0
23	27	7	1881.3157894736842	0.0
24	42	16	2333.6138613861385	0.0
25	32	12	1861.2755102040817	0.0
26	41	19	1587.6470588235295	0.0
27	28	9	1606.3766233766235	0.0
28	39	13	1618.7719298245613	0.0
29	39	15	1762.0176991150443	0.0
30	44	19	2232.9310344827586	0.0
31	33	11	1822.3592233009708	0.0
32	29	9	1697.107142857143	0.0
33	33	8	1616.1969696969697	0.0
34	46	11	1819.7721518987341	0.0
35	34	9	1743.8695652173913	0.0
36	26	6	1534.6206896551723	0.0
37	35	12	1613.883495145631	0.0
38	31	10	1529.6756756756756	0.0
39	41	14	1534.956043956044	0.0
40	33	8	1534.6808510638298	0.0

6.3.5. Agente con $\alpha = 0.5$, $\gamma = 0.9$ y $\varepsilon =$ decaying

Tiempo max. respuesta	Tasa max. error	Retraso episodio	Rango aprendizaje	Factor descuento
1500	0.1	2	0.5	0.5
Episodio	**Carga de trabajo**	**Paso episodio**	**Tiempo respuesta**	**Tasa error**
1	35	13	1546.7029702970297	0.0
2	37	16	1732.8085106382978	0.0
3	44	20	2498.28125	0.0
4	50	20	2282.046296296296	0.0
5	41	20	1807.7181818181818	0.0
6	33	14	1587.6666666666667	0.0
7	35	13	1862.3240740740741	0.0
8	36	14	1652.31	0.0
9	44	15	2410.567375886525	0.0
10	29	11	1730.2266666666667	0.0
11	40	19	2359.8888888888887	0.0
12	38	16	1684.2307692307693	0.0
13	51	20	1568.3727272727272	0.0
14	31	11	1614.8709677419354	0.0
15	40	16	1518.7851239669421	0.0
16	35	15	1736.5172413793102	0.0
17	33	11	1857.611111111111	0.0
18	44	15	1533.8977272727273	0.0
19	26	7	1500.9384615384615	0.0
20	34	9	2038.3676470588234	0.0
21	44	15	1765.035294117647	0.0
22	42	17	1576.4444444444443	0.0
23	38	16	1535.5689655172414	0.0
24	32	13	1524.83908045977	0.0
25	34	9	2046.6808510638298	0.0
26	35	10	1989.6938775510205	0.0
27	28	9	1614.6447368421052	0.0
28	41	15	1573.7652173913043	0.0
29	40	14	1512.0	0.0
30	42	13	1760.280991735537	0.0

Continúa en la siguiente página

Tabla 6.5 – continúa desde la página anterior

Episode	TotalWorkload	EpisodeStep	ResponseTime	ErrorRate
31	34	12	1771.141414141414	0.0
32	33	11	1858.0416666666667	0.0
33	32	7	2001.4555555555555	0.0
34	34	12	1904.909090909091	0.0
35	23	4	1545.047619047619	0.0
36	32	7	1851.7222222222222	0.0
37	30	8	1527.139534883721	0.0
38	34	9	1524.7857142857142	0.0
39	33	10	1894.8163265306123	0.0
40	32	12	1688.3877551020407	0.0

6.3.6. Agente con $\alpha = 0.1$, $\gamma = 0.5$ y $\varepsilon =$ decaying

Tiempo max. respuesta	Tasa max. error	Retraso episodio	Rango aprendizaje	Factor descuento
1500	0.1	2	0.5	0.5
Episodio	**Carga de trabajo**	**Paso episodio**	**Tiempo respuesta**	**Tasa error**
1	48	22	1735.28	0.0
2	43	18	1840.878048780488	0.0
3	39	17	1811.7614678899083	0.0
4	37	16	1929.27	0.0
5	41	16	1994.064	0.0
6	42	13	1641.4014598540145	0.0
7	40	19	1555.638888888889	0.0
8	37	16	1750.1714285714286	0.0
9	34	13	1825.3106796116506	0.0
10	34	15	1680.1770833333333	0.0
11	40	18	1813.811475409836	0.0
12	42	17	1635.5984251968505	0.0
13	33	8	1516.3084112149534	0.0
14	39	17	1677.954954954955	0.0

Continúa en la siguiente página

Tabla 6.6 – continúa desde la página anterior

Episode	TotalWorkload	EpisodeStep	ResponseTime	ErrorRate
15	28	9	1506.5692307692307	0.0
16	35	14	1877.926605504587	0.0
17	28	10	1594.0666666666666	0.0
18	38	15	1510.8403361344538	0.0
19	27	9	1558.4	0.0
20	30	8	1795.2553191489362	0.0
21	31	12	1979.7473684210527	0.0
22	35	10	1625.6788990825687	0.0
23	29	9	1729.3764705882354	0.0
24	31	9	1923.3030303030303	0.0
25	30	8	1883.6559139784947	0.0
26	22	4	1557.3548387096773	0.0
27	28	6	1857.1136363636363	0.0
28	32	10	1747.0485436893205	0.0
29	25	5	1595.7105263157894	0.0
30	33	11	1678.8076923076924	0.0
31	33	11	1659.326923076923	0.0
32	33	11	1633.3235294117646	0.0
33	31	9	1501.78125	0.0
34	33	8	1911.787037037037	0.0
35	40	14	1592.3	0.0
36	33	8	1606.9619047619049	0.0
37	32	7	1570.7596153846155	0.0
38	33	10	1851.4259259259259	0.0
39	33	11	1501.8446601941748	0.0
40	26	6	1766.6708860759493	0.0

6.3.7. Agente con $\alpha = 0.3$, $\gamma = 0.5$ y $\varepsilon =$ decaying

Tiempo max. respuesta	Tasa max. error	Retraso episodio	Rango aprendizaje	Factor descuento
1500	0.1	2	0.5	0.5
Episodio	**Carga de trabajo**	**Paso episodio**	**Tiempo respuesta**	**Tasa error**
1	34	14	1547.5666666666666	0.0
2	46	23	2016.3362068965516	0.0
3	29	10	1739.5	0.0
4	34	14	1572.1304347826087	0.0
5	62	26	2388.744	0.0
6	38	16	1913.7184466019417	0.0
7	44	18	1749.3023255813953	0.0
8	43	20	1739.5454545454545	0.0
9	44	19	1597.6302521008404	0.0
10	43	17	1615.128205128205	0.0
11	34	14	1576.8720930232557	0.0
12	46	17	1534.5692307692307	0.0
13	33	13	1865.0740740740741	0.0
14	43	14	1551.212765957447	0.0
15	42	16	2134.5481481481484	0.0
16	32	10	1524.5567010309278	0.0
17	42	15	1844.941605839416	0.0
18	42	16	2222.5	0.0
19	41	12	1748.9323308270677	0.0
20	31	11	1535.1460674157304	0.0
21	30	11	1552.641975308642	0.0
22	43	17	1544.4076923076923	0.0
23	29	9	1756.0	0.0
24	30	10	1683.5747126436781	0.0
25	44	14	1570.9402985074628	0.0
26	42	16	1506.0725806451612	0.0
27	42	13	1937.909090909091	0.0
28	40	15	1545.3070866141732	0.0
29	34	13	1505.735294117647	0.0
30	30	8	1628.2934782608695	0.0

Continúa en la siguiente página

Tabla 6.7 – continúa desde la página anterior

Episode	TotalWorkload	EpisodeStep	ResponseTime	ErrorRate
31	29	7	1612.5760869565217	0.0
32	26	6	1547.7532467532467	0.0
33	34	9	1725.2972972972973	0.0
34	34	9	1759.1160714285713	0.0
35	33	8	1725.8504672897195	0.0
36	29	8	1684.0747663551401	0.0
37	32	7	1722.048076923077	0.0
38	32	7	1887.5961538461538	0.0
39	35	11	1577.844827586207	0.0
40	27	7	1762.0119047619048	0.0

6.3.8. Agente con $\alpha = 0.7$, $\gamma = 0.5$ y $\varepsilon =$ decaying

Tiempo max. respuesta	Tasa max. error	Retraso episodio	Rango aprendizaje	Factor descuento
1500	0.1	2	0.5	0.5
Episodio	**Carga de trabajo**	**Paso episodio**	**Tiempo respuesta**	**Tasa error**
1	36	15	1519.9125	0.0
2	37	15	2015.9478260869564	0.0
3	29	11	1530.474358974359	0.0
4	52	23	1636.6940298507463	0.0
5	39	17	1839.3956043956043	0.0
6	44	15	1628.1818181818182	0.0
7	33	14	1862.8875	0.0
8	41	20	2086.660714285714	0.0
9	27	9	1541.5733333333333	0.0
10	39	18	2056.5238095238096	0.0
11	39	18	1521.142857142857	0.0
12	44	20	1572.9304347826087	0.0
13	42	19	1512.122807017544	0.0
14	33	13	1704.060975609756	0.0

Continúa en la siguiente página

Tabla 6.8 – continúa desde la página anterior

Episode	TotalWorkload	EpisodeStep	ResponseTime	ErrorRate
15	40	15	1514.5901639344263	0.0
16	44	20	1637.9193548387098	0.0
17	39	17	2262.105769230769	0.0
18	32	12	1631.2151898734178	0.0
19	47	20	1691.4770642201836	0.0
20	42	18	2155.659793814433	0.0
21	41	17	1537.87	0.0
22	38	13	2090.2803738317757	0.0
23	32	12	1640.5851063829787	0.0
24	44	14	1663.8175675675675	0.0
25	23	5	1510.126984126984	0.0
26	33	12	1684.9484536082475	0.0
27	40	14	2072.992366412214	0.0
28	27	7	1662.4698795180723	0.0
29	33	10	1769.504854368932	0.0
30	37	15	2347.733944954128	0.0
31	40	16	1576.3008849557523	0.0
32	38	12	1539.045045045045	0.0
33	27	7	1722.081081081081	0.0
34	30	8	1592.388888888889	0.0
35	24	5	1666.2957746478874	0.0
36	32	7	1656.1153846153845	0.0
37	26	7	1643.5	0.0
38	34	13	1948.4545454545455	0.0
39	41	16	1686.24	0.0
40	26	6	1553.139240506329	0.0

6.3.9. Agente con $\alpha = 0.9$, $\gamma = 0.5$ y $\varepsilon =$ decaying

Tiempo max. respuesta	Tasa max. error	Retraso episodio	Rango aprendizaje	Factor descuento
1500	0.1	2	0.5	0.5
Episodio	**Carga de trabajo**	**Paso episodio**	**Tiempo respuesta**	**Tasa error**
1	45	21	1536.8918918918919	0.0
2	38	18	1883.25	0.0
3	34	13	1804.4	0.0
4	32	13	1514.1954022988505	0.0
5	38	17	1524.904761904762	0.0
6	53	24	2221.1804511278197	0.0
7	42	19	1560.4684684684685	0.0
8	30	12	1589.6707317073171	0.0
9	34	14	2017.6734693877552	0.0
10	30	10	1725.3483146067415	0.0
11	37	17	2081.9504950495048	0.0
12	32	13	1718.5795454545455	0.0
13	32	13	1773.375	0.0
14	42	18	1631.512	0.0
15	43	17	2265.839285714286	0.0
16	45	15	2447.5423728813557	0.0
17	41	15	1669.3636363636363	0.0
18	40	17	1510.5641025641025	0.0
19	33	14	1681.154761904762	0.0
20	24	6	1591.603448275862	0.0
21	38	14	2124.3978494623657	0.0
22	50	23	1590.5309734513273	0.0
23	40	19	1512.9642857142858	0.0
24	36	14	1986.8349514563106	0.0
25	34	14	1905.6	0.0
26	27	9	1524.6666666666667	0.0
27	26	8	1510.391304347826	0.0
28	27	9	1558.625	0.0
29	31	11	1520.25	0.0
30	37	16	1522.468085106383	0.0

Continúa en la siguiente página

Tabla 6.9 – continúa desde la página anterior

Episode	TotalWorkload	EpisodeStep	ResponseTime	ErrorRate
31	31	12	1691.1818181818182	0.0
32	41	12	1581.0	0.0
33	30	10	1652.1555555555556	0.0
34	28	8	1851.6091954022988	0.0
35	32	12	2113.0	0.0
36	33	12	1530.0729166666667	0.0
37	28	6	1993.465909090909	0.0
38	32	10	1548.357142857143	0.0
39	29	9	1628.9176470588236	0.0
40	34	12	1701.0526315789473	0.0

6.3.10. Agente con $\alpha = 0.5$, $\gamma = 0.5$ y $\varepsilon = 0.2$

Tiempo max. respuesta	Tasa max. error	Retraso episodio	Rango aprendizaje	Factor descuento
1500	0.1	2	0.5	0.5
Episodio	**Carga de trabajo**	**Paso episodio**	**Tiempo respuesta**	**Tasa error**
1	68	14	1508.173076923077	0.0
2	45	16	1735.837837837838	0.0
3	40	14	1946.4	0.0
4	41	12	1542.871794871795	0.0
5	40	13	1523.672268907563	0.0
6	31	9	1536.6511627906978	0.0
7	41	12	2500.008474576271	0.0
8	41	12	1525.4434782608696	0.0
9	44	19	1536.9166666666667	0.0
10	35	15	1578.8214285714287	0.0
11	34	13	1550.4137931034484	0.0
12	40	13	1541.0238095238096	0.0
13	38	12	1755.209090909091	0.0
14	38	9	2036.4453125	0.0

Continúa en la siguiente página

Tabla 6.10 – continúa desde la página anterior

Episode	TotalWorkload	EpisodeStep	ResponseTime	ErrorRate
15	30	10	2713.511111111111	0.0
16	28	8	1556.698795180723	0.0
17	41	17	1504.8823529411766	0.0
18	29	9	2216.777777777778	0.0
19	40	11	1519.4700854700855	0.0
20	42	16	1622.976	0.0
21	38	14	1742.9495798319329	0.0
22	41	17	1637.8790322580646	0.0
23	34	11	1965.1149425287356	0.0
24	34	9	1574.0816326530612	0.0
25	25	6	1509.8805970149253	0.0
26	52	16	1585.9029126213593	0.0
27	34	13	1544.7764705882353	0.0
28	35	14	1668.8285714285714	0.0
29	41	17	2143.1732283464567	0.0
30	34	13	2188.048076923077	0.0
31	24	6	1585.485294117647	0.0
32	33	8	1809.1028037383178	0.0
33	30	10	1558.1868131868132	0.0
34	28	9	1553.3902439024391	0.0
35	35	10	1615.796460176991	0.0
36	40	11	1908.7751937984497	0.0
37	33	8	1727.057142857143	0.0
38	34	12	1566.5377358490566	0.0
39	37	13	1541.5128205128206	0.0
40	32	12	1707.908163265306	0.0

6.3.11. Agente con $\alpha = 0.5$, $\gamma = 0.5$ y $\varepsilon = 0.8$

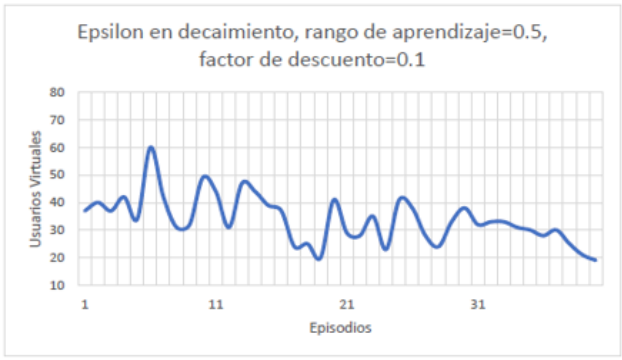
Tiempo max. respuesta	Tasa max. error	Retraso episodio	Rango aprendizaje	Factor descuento
1500	0.1	2	0.5	0.5
Episodio	**Carga de trabajo**	**Paso episodio**	**Tiempo respuesta**	**Tasa error**
1	46	20	2188.919117647059	0.0
2	45	21	2035.398148148148	0.0
3	29	11	1695.9493670886077	0.0
4	49	21	2419.4727272727273	0.0
5	46	20	1912.5925925925926	0.0
6	36	17	1611.8	0.0
7	34	14	1951.9204545454545	0.0
8	31	12	1686.2134831460673	0.0
9	40	17	1833.880733944954	0.0
10	33	14	1665.0238095238096	0.0
11	34	15	1881.741935483871	0.0
12	23	5	1515.5737704918033	0.0
13	22	4	1531.8524590163934	0.0
14	24	6	1732.9516129032259	0.0
15	40	19	2156.8818181818183	0.0
16	34	13	1611.5	0.0
17	31	12	1885.8117647058823	0.0
18	53	24	1606.2268907563025	0.0
19	44	21	1627.5416666666667	0.0
20	30	11	1738.7349397590363	0.0
21	43	18	1531.2673267326732	0.0
22	44	20	1557.945945945946	0.0
23	29	11	1557.32	0.0
24	30	12	1729.5641025641025	0.0
25	36	14	2144.175925925926	0.0185
26	20	2	1927.301886792453	0.0
27	44	20	1926.8245614035088	0.0
28	39	16	1537.5294117647059	0.0
29	44	20	2204.7295081967213	0.0
30	33	13	2053.2474226804125	0.0

Continúa en la siguiente página

Tabla 6.11 – continúa desde la página anterior

Episode	TotalWorkload	EpisodeStep	ResponseTime	ErrorRate
31	27	9	1601.5	0.0
32	42	19	1622.2622950819673	0.0
33	31	12	1698.1704545454545	0.0
34	53	21	2582.346405228758	0.0
35	30	12	1792.3291139240507	0.0
36	34	14	1741.4148936170213	0.0
37	34	16	1504.4157303370787	0.0
38	50	22	1532.1339285714287	0.0
39	34	15	1534.0549450549452	0.0
40	32	12	1521.6444444444444	0.0

6.4. Resultados de los experimentos de eficiencia del agente




Epsilon en decaimiento, rango de aprendizaje=0.3, factor de descuento=0.5
Usuarios Virtuales
Episodios


Epsilon en decaimiento, rango de aprendizaje=0.7, factor de descuento=0.5
Usuarios Virtuales
Episodios


Epsilon en decaimiento, rango de aprendizaje=0.9, factor de descuento=0.5
Usuarios Virtuales
Episodios


Epsilon=0.2, rango de aprendizaje=0.5, factor de descuento=0.5
Usuarios Virtuales
Episodios


Epsilon=0.8, rango de aprendizaje=0.5, factor de descuento=0.5
Usuarios Virtuales
Episodios


Linea base estandar
Usuarios Virtuales
Episodios