# Dónde estamos? | Web de la Escuela de Ingeniería Informática de la ...

**Trabajo de Fin de Título en Ingeniería Informática**

Generador Sintético de datos para una aplicación de Fintech: Finbook

Juan Alberto Ureña Rodriguez

Tutor: José Juan Hernández Cabrera

Fecha: 1 de Julio de 2020

Contenido

[ 1](#_Toc44349469)

[**1. Explicación del Ámbito del Proyecto** 3](#_Toc44349470)

[**1.1 Introducción del Ámbito** 3](#_Toc44349471)

[**1.1.1 Hacia la Digitalización** 3](#_Toc44349472)

[**1.1.2 Liquidación Automatizada del IVA** 5](#_Toc44349473)

[**1.2 Factura** 6](#_Toc44349474)

[**1.2.1 Factura Genérica** 6](#_Toc44349475)

[**1.2.2 Factura Electrónica en México** 7](#_Toc44349476)

[**1.3 Explicación de Aplicación “Fintech”** 8](#_Toc44349477)

[**2. Motivación y Objetivos Iniciales** 9](#_Toc44349478)

[**2.1 Motivación** 9](#_Toc44349479)

[**2.2 Objetivos Iniciales** 10](#_Toc44349480)

[**3. Desarrollo** 11](#_Toc44349481)

[**3.1 Metodología Aplicada** 11](#_Toc44349482)

[**3.2 Planificación** 12](#_Toc44349483)

[**4. Herramientas Usadas** 14](#_Toc44349484)

[**4.1 Herramientas de Hardware** 14](#_Toc44349485)

[**4.2 Herramientas de Software** 14](#_Toc44349486)

[**5. Arquitectura** 15](#_Toc44349487)

[**5.1 Arquitectura Publisher/Subscriber** 15](#_Toc44349488)

[**5.2 Arquitectura del Módulo desarrollado** 15](#_Toc44349489)

[**5.2.1 Arquitectura Multi-Agente** 15](#_Toc44349490)

[**5.2.2 Estructura Interna** 15](#_Toc44349491)

[**6. Competencias Especificas Cubiertas** 15](#_Toc44349492)

[**7. Generador Sintético de datos** 16](#_Toc44349493)

[**7.1 Introducción** 16](#_Toc44349494)

[**7.2 Simulación** 17](#_Toc44349495)

[**7.2.1 Ciclo de Vida de la Simulación** 17](#_Toc44349496)

[**7.2.2 Agentes Activos o Simulables** 19](#_Toc44349497)

[**7.2.3 Bajas y Altas de Simulables** 33](#_Toc44349498)

[**7.2.4 Administración** 34](#_Toc44349499)

[**7.2.5 Facturas** 58](#_Toc44349500)

[**7.2.6 Eventos** 61](#_Toc44349501)

[**7.2.7 Capa de Datos** 63](#_Toc44349502)

[**7.2.8 Preparación de Simulables** 77](#_Toc44349503)

[**7.2.9 Resumen** 77](#_Toc44349504)

[**7.3 Entorno de usuario de la Simulación** 78](#_Toc44349505)

[**7.3.1 Introducción al entorno** 78](#_Toc44349506)

[**7.3.2 Servidor** 79](#_Toc44349507)

[**7.3.3 Cliente** 82](#_Toc44349508)

[**8. Conclusiones** 82](#_Toc44349509)

[**8.1 Resultado Obtenido** 82](#_Toc44349510)

[**8.1.1 Del Proyecto** 82](#_Toc44349511)

[**8.1.2 A Nivel Personal** 83](#_Toc44349512)

[**8.1.3 A Nivel Laboral** 83](#_Toc44349513)

[**8.2 Aportaciones** 83](#_Toc44349514)

[**8.3 Trabajos Futuros** 84](#_Toc44349515)

[**9. Bibliografía** 85](#_Toc44349516)

# **1. Explicación del Ámbito del Proyecto**

## **1.1 Introducción del Ámbito**

### **1.1.1 Hacia la Digitalización**

En la economía actual, las facturas se generan principalmente en papel en la mayoría de los sectores y países, esto se debe a que es el método más usado durante las últimas décadas. En el siglo XX y sobre todo antes de la creación del ordenador y las redes de internet, era la mejor opción para generar facturas, sin embargo, ¿sigue siendo actualmente la manera óptima?, ¿o existen nuevas posibilidades para gracias a las nuevas tecnologías? Esa es una pregunta que se responderá durante la presentación de este trabajo.

Como respuesta se ha originado un nuevo tipo de factura, la factura electrónica. Esta factura al contrario de la común factura en papel se crea, se guarda y se distribuye virtualmente lo que permite una mayor facilidad y seguridad en mundo socioeconómico actual, como veremos a continuación. La factura electrónica funciona como cualquier fichero o archivo de datos, guarda información, pero en este caso guarda los datos de una factura. El formato de la factura electrónica más extendido es XML (eXtensible Markup Language) debido a su simplicidad y la claridad a la hora de leer una factura por una persona que conozca un poco el formato o para un programa que la procese. Este uso de formato genérico permite como acabo de nombrar, una “generalización” de la factura, es decir, permite o más bien obliga tener un conjunto de reglas rígido que seguir para la creación de las facturas formando una estructura común y constante entre todas ellas y por lo tanto una mayor facilidad a la hora de leerlas por una persona o procesarlas por un ordenador.

Esta ventaja de la factura electrónica ayuda enormemente a sectores como la fiscalía, hacienda o la industria, así como cualquier empresa de servicios o venta de productos, es decir cualquier entidad que genere facturas, debido a su facilidad de creación, distribución y uso de estas. Además, otra ventaja que tiene la factura electrónica es la seguridad que trae de por si el uso de estas ya que controla el flujo de facturas generados virtualmente y permite controlar las evasiones fiscales que trae consigo las facturas en papel que nunca son declaradas.

Por último, tenemos la ventaja que se va a aprovechar y demostrar en este proyecto, la capacidad de procesar y analizar estas facturas. Esto se debe como ya nombramos anteriormente, al formato que usa, que favorece el procesamiento y el análisis de estas ya que la estructura es única por lo que todas las facturas se pueden leer y obtener datos de la misma manera, de forma iterativa y constante de todas y finalmente generar un reporte de todas estas mucho más rápido que una persona experta en esto. Este sector que automatiza el análisis de datos se denomina Big Data, que como su nombre indica, permita analizar grandes volúmenes de datos de forma rápida y eficaz en todo tipo de sectores, uno de ellos la facturación. Esta es una de las grandes ventajas de la factura electrónica con respecto a la de papel, y es el Big Data el futuro en muchos sectores que trabajen con grandes volúmenes de datos.

La factura electrónica se está empezando a extender actualmente y algunos países como México, Chile, Brasil e Italia están empezando a usarlas como en todos los sectores del país. En algunos como México están propiciadas por tener un mayor control de estas y evitar robos de dinero y fraudes fiscales. Este número de países sigue aumentando durante estos años y se espera que sea la factura más usada y extendida en el futuro, como tantos otros avances que ha propiciado la ciencia y la tecnología.

### **1.1.2 Liquidación Automatizada del IVA**

El Impuesto de valor añadido es un tributo que grava la entrega de bienes y las prestaciones de servicios. La liquidación del IVA se ha de realizar trimestralmente, si bien las grandes empresas lo realizarán mensualmente. El modelo fiscal que hay presentar es el modelo 303 y se presenta telemáticamente a la Agencia Tributaria. Cada entrega de bienes y servicios estará gravada sobre el impuesto sobre valor añadido, de tal manera que la empresa cuando venda sus mercancías deberá consignar en la factura el importe del IVA repercutido. De otro lado, cuando la empresa compre bienes y servicios, el proveedor le entregará una factura por el importe gravado por el IVA, denominado IVA soportado. La base imponible del IVA se determina por el precio facturado menos todos los descuentos. A dicho importe se le aplicará un tipo de gravamen el cual dependerá del tipo de operación.

La liquidación del IVA consiste en determinar la diferencia entre el IVA repercutido y el IVA soportado. Tal diferencia, si es positiva, deberá ingresarse en la Agencia Tributaria a través del referido modelo 303. Si la diferencia es negativa, se generará un derecho de cobro que se compensará en la siguiente liquidación, a menos que nos encontremos al cierre del ejercicio económico.

La información fiscal que se desprende de las operaciones económicas que vienen gravadas por el IVA también se representan en la contabilidad, el IVA soportado que es deducible del IVA repercutido, es decir, que minora la base imponible del impuesto vendrá representado en una cuenta denominada hacienda pública, IVA soportado del grupo 4 del cuadro de cuentas del plan general de contabilidad. Cuando el IVA soportado no puede deducirse del IVA repercutido, formará parte del precio de adquisición y por tanto de la cuenta de gasto o de activo. Ello significa que la empresa al no poder reducir el IVA soportado no tendrá ninguna cuenta específica en los estados financieros. Además, los comerciantes minoristas que sean personas fijas pueden acogerse al recargo de equivalencia que supone pagar un IVA más alto a cambio de no tener que presentar declaraciones del IVA a hacienda, de esta forma el IVA soportado se paga directamente por sus compras al proveedor y se simplifica la gestión porque no hay que llevar libros de IVA ni guardar facturas. Esto se relaciona con sector de la restauración ya que es un negocio minorista y es el sector que principalmente se va a enfocar mi Trabajo de Fin de Grado.

Tras una breve introducción de la liquidación, llega unos de los enfoques del Big Data en el sector fiscal, la liquidación automática del IVA, que consiste en el análisis y cálculo de IVA de repercutido y el IVA soportado de forma automática, rápida y eficaz del precio facturado de una empresa o de cualquier entidad económica que ofrezca bienes o servicios. Además de suponer una enorme facilidad para la Agencias tributarias a la hora comprobar esos datos, supone un ahorro de tiempo a la hora de presentar el modelo fiscal 303 ante la agencia tributaria telemáticamente.

Este avance permite un ahorro económico importante en las empresas y una facilidad para estas a la hora de hacer las desgravaciones ya que nos ahorramos “la mano de obra” que se encarga de realizar tal análisis.

## **1.2 Factura**

### **1.2.1 Factura Genérica**

Una factura es un documento mercantil que refleja toda la información de una operación de compraventa. El sistema de facturación está basado en dos tipos de facturas:

* Factura completa u ordinaria, que posee los siguientes campos
* Factura simplificada: Las facturas simplificadas, que sustituyen a los tiques, tienen un contenido más reducido que las facturas completas y por lo general, están permitidas solo para facturas no superiores a unos 400 euros.

La factura completa tiene los siguientes campos:

* Número y, en su caso, serie. La numeración ha de ser correlativa dentro de cada serie.
* Fecha de su expedición.
* Fecha de realización de la operación documentada en la factura (si no coincide con la fecha de expedición).
* Número de identificación fiscal (NIF), nombre y apellidos o denominación social tanto del obligado a expedir la factura como del destinatario de las operaciones.
* Domicilio del expedidor y del destinatario.
* Descripción de las operaciones y todos los datos necesarios para la determinación de la base imponible y su importe y precio unitario sin impuesto, así como cualquier descuento o rebaja que no esté incluido en el precio unitario.
* Tipos impositivos aplicados a las operaciones.
* Cuota tributaria, que deberá consignarse por separado.
* Sede de actividad o establecimiento si alguna de las partes o ambas disponen de varias sedes fijas del negocio.
* Moneda
* Lengua, Las facturas pueden expedirse en cualquier lengua. Existe la posibilidad de exigir traducción al castellano u otra lengua oficial.
* Firma y localidad (opcional).
* Medios de expedición, papel o formato electrónico que permita garantizar la autenticidad de su origen, la integridad de su contenido y su legibilidad.

También existe la denominada factura proforma que es simplemente un anticipo de la posterior factura real o la factura recapitulativa que se genera por periodos mensuales, es decir, pueden incluirse en una sola factura las operaciones realizadas en distintas fechas para un mismo destinatario. Deben ser expedidas, como máximo, el último día del mes natural en el que se realizaron las operaciones documentadas en la factura.

La factura electrónica es el documento electrónico equivalente a la factura en papel. Consiste en una transmisión telemática entre un emisor y un receptor por medio de dispositivos electrónicos capaces de generar las facturas adecuadas, que será la que estudiaremos más a fondo en el caso de México.

Las entidades o empresas deberán conservar los siguientes documentos:

• Las facturas recibidas.

• Las copias de las facturas expedidas.

La obligación de conservación puede cumplirse mediante la utilización de medios electrónicos. Las obligaciones se pueden cumplir materialmente por un tercero que actúe en nombre de la empresa. El Código de Comercio obliga a los empresarios a conservar los libros contables, la correspondencia, la documentación y los justificantes de sus operaciones durante seis años, a partir del último asiento realizado en los libros.

### **1.2.2 Factura Electrónica en México**

La factura electrónica en México es ya una realidad en este país, que debido a varias causas de índole fiscal y económica ha traído consigo indirectamente un gran cambio en la manera de crear y administrar las facturas.

La facturación electrónica en México se inició en el 2004, cuando el Servicio de Administración Tributaria (SAT) creó el marco legal que definió la implantación del Comprobante Fiscal Digital a través de Internet (CDFI). Con el tiempo se ha convertido en uso obligatorio para la contabilidad electrónica de México y actualmente es utilizado en el 100% de las transacciones económicas. El éxito de este sistema implementado en México ha hecho que otros gobiernos de Latinoamérica estén siguiendo sus pasos y trabajan en la implantación de una facturación electrónica propia. El sistema de facturación electrónica de México ha destacado gracias a su rápida difusión, que le ha permitido extenderse en pocos años por toda la sociedad mexicana.

El CFDI es un documento XML el cual está diseñado para cumplir con la especificación y exigencias del SAT y son las siglas que indican 4 características importantes de esta factura:

* Comprobante: justifica ante la administración fiscal que realmente se efectuó un proceso de compra venta y que se pagaron los impuestos designados.
* Fiscal: garantiza que el comprobante se encuentra en el marco legal, que tiene que ajustarse a la legislación aplicable y enviarse en plazo al SAT. Este punto es muy importante ya que era una las causas más importantes que motivaban este cambio en la facturación debido a los fraudes fiscales y la evasión de impuestos.
* Digital: las facturas están creadas en un sistema binario y son almacenadas en un disco duro donde se puede hacer una búsqueda de una factura, de un archivo y encontrarla en segundos tan solo indicando su nombre, fecha o alguna otra característica de esta. Además, se puede mantener archivada fácilmente sin llegar a perderla en ningún momento.
* Internet: Las empresas que emiten CFDI, las emiten con su propio sistema de facturación lo que genera una Descentralizar la facturación y, por lo tanto, se ahorra un gran capital en términos de seguridad informática. Por lo tanto, este sistema permite emitir CFDI desde todo tipo de dispositivos electrónicos como un celular en un restaurante o en una venta hecha fuera de las oficinas.

La Plataforma Interna del SAT es la encargada de coordinar, administrar y sincronizar a todos los proveedores que están autorizados, también se encarga de administrar la información concerniente a cada uno de los contribuyentes y sus respectivas facturas electrónicas asociadas. Poseen un buen sistema de seguridad basado en el sistema de criptografía de clave pública, por lo que le otorga a cada contribuyente algunas claves, una pública y otra privada.

Como conclusión de este sistema, podemos ver que tenemos un sistema bien desarrollado y con vistas de futuro que con el tiempo el resto de los países seguirá y posiblemente se convierta en la base de la facturación en el mundo.

## **1.3 Explicación de Aplicación “Fintech”**

Tecnología Financiera o Fintech es una industria que aplica las nuevas tecnologías de la información, concretamente el creciente sector del Big Data, al sector financiero. Esta industria este dividida en cuatro segmentos:

* Herramientas de operación y medios de pago
* Conocimiento del cliente y Big Data
* Seguridad e identificación de personas
* Dinero electrónico

Las aplicaciones de Tecnología financiera (o Aplicación Fintech) es un tipo de aplicación financiera que proporciona herramientas tecnológicas a este sector para poder facilitar la labor de los usuarios de la aplicación a la hora de tratar con grandes volúmenes de datos.

Estas aplicaciones tienen como objetivo automatizar procesos que hoy en día se producen manualmente como la facturación, mediante la digitalización de los datos y el procesamiento automático de estos por parte de procesadores y células inteligentes. Para ello, esta entidad inteligente tiene la necesidad de poder procesar lenguaje natural y derivados, o leer archivos usados para el almacenamiento de datos, como XML o JSON, para poder realizar adecuadamente su labor, esto suele llevar detrás el desarrollo de una inteligencia que pueda entender tales lenguajes. Esta tecnología tiene diversos como:

* Pagos y remesas.
* Préstamos.
* Gestión de finanzas empresariales.
* Gestión de finanzas personales.
* Crowdfunding (financiamiento de proyectos).
* Gestión de inversiones.
* Seguros.
* Educación financiera y ahorro.
* Soluciones de scoring, identidad y fraude.
* Trading y mercados.

# **2. Motivación y Objetivos Iniciales**

## **2.1 Motivación**

Este proyecto ha supuesto un enorme interés desde el inicio del desarrollo del mismo, debido a varios factores, principalmente por el hecho de crear por primera vez una aplicación Fintech, un campo completamente novedoso para mí, siguiendo una arquitectura de Big Data, otro sector completamente nuevo que, aplicado al sector financiero, supuso estudiar un nuevo enfoque de la programación que conllevaría aprender mucha información nueva para el desarrollo del proyecto.

Otro aspecto que destacar es el tamaño del proyecto completo, que ha sido un nuevo reto para mí, porque el haber desarrollado un módulo de este proyecto ha llevado mucho tiempo, además de la posterior integración con el resto de mis compañeros. Es sin duda el proyecto más ambicioso que he realizado hasta el momento y al que le he dedicado mucho esfuerzo y dedicación con entusiasmo y emoción.

Centrándonos en mi proyecto personal, el hecho de realizar una simulación en la que los elementos de este interactúen entre si realizando unas acciones realistas y aceptables dentro del marco de la normalidad, era de antemano, un reto, el cual me motivó enormemente ya que el poder hacer algo tan avanzado individualmente y solo con la ayuda de mi tutor era una idea que parecía imposible, debido a mi nivel inicial al empezar el proyecto.

Otra motivación proviene de poder realizar este proyecto partiendo desde cero, sin ningún tipo de trabajo ya hecho, teniendo yo la responsabilidad de la arquitectura y las dependencias a usar, realizando todas las funcionalidades necesarias de la simulación que se necesitaba. Además, siguiendo con las ideas del profesor, quería un proyecto con el mínimo de dependencias externas, ya que son factores que luego pueden afectar directamente al proyecto, por lo tanto, todos los distintos módulos y funcionalidades que iba a necesitar en el futuro fueran implementadas, en la medida de lo posible, todas por mí. Sin embargo, tenía claro que quería buscar un equilibrio en el que también se puedan ciertas librerías ya existentes que no supongan grandes dependencias en mi proyecto final.

## **2.2 Objetivos Iniciales**

Este proyecto tiene como propósito generar un Publisher (feeder) de la Plataforma de Datos, es decir, un generador de datos financieros realistas con el fin de crear un generador de datos sintéticos que aporte tanto facturas de clientes, como del personal o los proveedores de materias primas involucrados en el desarrollo económico de las empresas.

Estas empresas o agentes pondrán a disposición un gran volumen datos financieros que serán utilizados por otros proyectos posteriormente, para su estudio y análisis. Este objetivo será enfocado al ámbito de la restauración, y será posible a través de un ciclo iterativo basado en tres fases, Modelación, Simulación y Validación:

Modelación: Consiste en estructurar y abstraer los datos, elementos y eventos de la realidad para poder simularlos posteriormente, ya sea el consumo de restaurantes por parte de clientes y las facturas generadas, además de las nóminas y la compra de materias primas.

Simulación: añadir estos elementos dentro de la simulación y que interactúen entre sí mediante compra/venta de materias primas, contratación/despido de personal y consumo de los clientes.

Validación: Análisis de los datos recibidos de la simulación y comprobación de que lo datos recibidos son acordes y se reflejan en la realidad.

A partir de ahí, han surgido nuevos requisitos que cumplir, como la generación de facturas de servicios, de devolución de productos, además de añadir suficientes elementos a la simulación, como contratos de trabajadores, nacimiento y muerte de individuos y empresas, jubilación de los trabajadores, pago de impuestos por parte de las empresas, pago de hipoteca y concluyendo con aspecto muy importante en las simulaciones, como es la toma de decisiones adecuadas según la situación, por parte de los individuos y empresas de la simulación, todo esto para mejorar el realismo de la simulación.

# **3. Desarrollo**

## **3.1 Metodología Aplicada**

El lenguaje de programación que utilizaremos principalmente para elaborar la aplicación “FinTech” será Java, utilizando el IDE “IntelliJ”. Se hará uso de la herramienta “Intino”, que permite la creación de la plataforma de datos. Se usará una arquitectura de programación funcional, aplicando un desarrollo guiado por pruebas (TDD – Test-Driven Development) e iterativo, en el que se establecerán una serie de objetivos/hitos a cumplir en cada iteración. Cada iteración tendrá una duración aproximada de 2 semanas, en la que se le dedicará un total de 50 horas en cada una.

Se hará uso de un “bus” de mensajería proporcionado por “Intino” para comunicar las distintas partes funcionales del proyecto realizadas por los distintos participantes. Cabe destacar que “Intino” usa la tecnología JMS (Java Messaging Service) que consiste en una solución para el uso de colas de mensajes: es un estándar de mensajería que permite a los componentes de aplicaciones crear, enviar, recibir, y leer mensajes, y que permite una comunicación confiable de manera asíncrona.

Para el control de versiones del proyecto se utilizará la tecnología “Git”, y dada la naturaleza grupal del proyecto, se aprovechará el modelo de ramificación “GitFlow” para facilitar el desarrollo concurrente y organizado por parte de todos los integrantes del proyecto.

## **3.2 Planificación**

He dividido mi proyecto en 3 fases, estudio del ámbito y las tecnologías que se usaran, desarrollo e implementación del proyecto, y validación y prueba del proyecto en el entorno de usuario. Por último, se encuentra la documentación final.

|  |  |  |
| --- | --- | --- |
| ***Fases*** | ***Duración Estimada (horas)*** | ***Tareas***  ***(nombre y descripción, obligatorio al menos una por fase)*** |
| Primera Iteración: Estudio previo y Análisis de las tecnologías y la metodología que se usaran. | 50 | Iteración 1.1: Estudio y búsqueda de un generador de personas aleatorias usadas como clientes de la simulación |
| Iteración 1.2: Estudio de la tecnología JSOUP para la obtención de datos de restaurantes la página web TripAdvisor, usados como restaurantes de la simulación. |
| Iteración 1.3: Análisis de datos relacionados con la parametrización de la simulación. Por ejemplo, distribuciones de sueldos, del tamaño del restaurante, del grado de consumo de restaurantes por parte de la población y el número de personas que son invitadas en función de sus relaciones sociales. |
| Iteración 1.4: Estudio de las facturas electrónicas generadas en México usando CFDI, que es un documento XML que cumple con las especificaciones proporcionadas por la SAT (Servicio de Administración Tributaria). |
| Segunda Iteración: Diseño, Desarrollo e Implementación de la Simulación | 250 | Iteración 2.1: Desarrollo de un módulo que extraiga de internet, una lista de restaurantes con sus datos para ser usados posteriormente en la simulación. |
| Iteración 2.2: Desarrollo de un módulo que extraiga de un archivo CSV, una lista de clientes y empresas secundarias con sus datos para ser usados posteriormente en la simulación. |
| Iteración 2.3: Guardar los datos de clientes, empresas secundarias y restaurantes en una base de datos fácilmente accesible para su rápida lectura durante la simulación. |
| Iteración 2.4: Diseño y desarrollo de un sistema de tiempo que permita controlar un calendario artificial con días meses y años, en el que sucederán todos los actos dentro de la simulación. |
| Iteración 2.5: Diseño y desarrollo de un sistema de simulables que interactuarán independientemente cada día de la línea de tiempo, todos los elementos que formen parte de la simulación serán simulables. |
| Iteración 2.6: Diseño e implementación de un sistema de consumo de restaurantes por parte de los clientes con cierta temporalidad parametrizada, que genere facturas de comidas y que sean incluidas en sus datos financieros para su posterior análisis, esto generará facturas de compras. |
| Iteración 2.7: Adición de empresas relacionadas con el sector de la Restauración como proveedores y empresas de servicios que provean de materias primas y servicios a los restaurantes, esto generará facturas de compras de productos, servicios y devoluciones. |
| Iteración 2.8: Desarrollo de un sistema de contrataciones de los trabajadores que trabajaran en los restaurantes a cambio de un salario que influya también en los datos financieros de la empresa, esto generará facturas de nóminas. |
| Iteración 2.9: Diseño e implementación de un sistema de estrategias a la hora de tomar decisiones en las distintas situaciones que se presentan. Por ejemplo, a la hora de contratar los trabajadores, los restaurantes elegirán quien contratar según la situación. |
| Iteración 2.10: Desarrollo de un sistema de contrataciones de los trabajadores que trabajaran en los restaurantes a cambio de un salario que influya también en los datos financieros de la empresa, esto generará facturas de nóminas. |
| Iteración 2.11: Implementación de un sistema de impuestos y pagos de hipoteca del local por parte de las empresas. |
| Iteración 2.12: Creación de una vista en web para mostrar la simulación en la que el usuario puede ver los eventos que se producen, e interactuar y cambiar los ajustes de este. |
| Tercera Iteración: Evaluación, Validación y Prueba de la Simulación en el Entorno de Finbook | 60 | Iteración 3.1: Testeo y búsqueda de errores. |
| Iteración 3.2: Integración del Publisher con la Plataforma de datos |
| Cuarta Iteración Documentación y Presentación del Proyecto realizado | 40 | Iteración 4.1: Documentar todo el proyecto realizado. |
| Iteración 4.2: Preparación de la presentación del trabajo. |

# **4. Herramientas Usadas**

## **4.1 Herramientas de Hardware**

* Ordenador de Sobremesa: herramienta fundamental y principal para el desarrollo de este proyecto, ya que ha sido el lugar de creación, desarrollo y almacenamiento de este, además del almacenamiento y uso de todos los programas necesarios para el desarrollo de la simulación.
* Ordenador Portátil: herramienta adicional para continuar con el desarrollo de este proyecto cuando no me encuentro cerca del lugar de trabajo principal.
* Conexión a internet: herramienta esencial en todo tipo de proyectos hoy en día, para la documentación y la descarga de aplicaciones necesarias.

## **4.2 Herramientas de Software**

* J**ava:** lenguaje de programación principal usado durante el desarrollo y la implementación de la simulación.
* **Javascript + JSP + CSS:** lenguajes y herramientas principales en el entorno gráfico y visual del usuario, ya que he usado el entorno web para mostrar la simulación.
* **Intellij IDEA:** programa principal de desarrollo del proyecto tanto en la parte de servidor con Java como la parte del cliente en web.
* **Tomcat Server:** herramienta de Tomcat que he usado como servidor del proyecto.
* **Javax Web-socket:** herramienta de Javax para el desarrollo de los sockets entre el servidor y el cliente con el pasar información directa entre archivos de Javascript y archivos Java.
* **TripAdvisor:** página web usada para leer los datos de todos los restaurantes activos en Gran canaria, a través de la librería Jsoup. (9)
* **Jsoup:** librería para “Web Scrapping”, es decir, leer datos de los archivos html de las páginas web. (5)
* **Generador de datos aleatorios:** he usado un generador de datos para crear los perfiles de las distintas empresas y personas de la simulación, exceptuando los restaurantes. (6)
* **Jquery:** librería para complementar en Javascript en la administración de la web. (8)
* **DB Browser for SQLite:** programa para administrar leer, y guardar los datos en bases de datos SQL que se manejaran durante la simulación, esto incluyen datos de clientes, empresas secundarias, facturas y restaurantes. (10)

# **5. Arquitectura**

## **5.1 Arquitectura** **Publisher/****Subscriber**

Esta arquitectura es muy usada para el desarrollo de proyectos Big Data y es la arquitectura elegida para este proyecto de Finbook (14). La arquitectura está dividida en tres módulos: los Publisher (Publicadores), los Subscriber (Suscriptores) y el Data Hub (Centro de datos). A continuación, se explicarán las funciones de cada módulo:

* Centro de datos: como su nombre indica, es el centro donde se guardan los datos. Se encarga de recibir los datos de los publicadores y proveer de esos datos a los suscriptores. Cuando se quiere publicar un dato, genera un evento indicando los datos del envío y el dato a guardar. Luego manda un evento a los suscriptores de que se ha añadido un nuevo dato. Por último, el suscriptor obtiene el dato de ese evento para su posterior uso.
* Publicador: es un módulo que genera datos para que otros módulos puedan usarlos, puede haber varios publicadores conectados al centro de datos. A través de un módulo de conexión perteneciente al centro de datos, se conecta al servidor donde se encuentra desplegado el centro de datos, el módulo genera el dato, y el módulo de publicador genera el evento en el centro de datos.
* Suscriptor: este módulo es el encargado de recibir esos datos para su posterior uso, puede haber varios suscriptores suscritos al centro de datos. A través del módulo desarrollado como suscriptor del centro de datos, se conecta al servidor al igual que el publicador, pero para suscribirse al centro de datos. Así, cuando se genere algún evento, el centro de datos publicará el evento a todos los suscriptores, usando el módulo de conexión, y de este obteniendo el dato en cuestión.

A continuación, se muestra visualmente como es una arquitectura Publisher/Subscriber genérica.

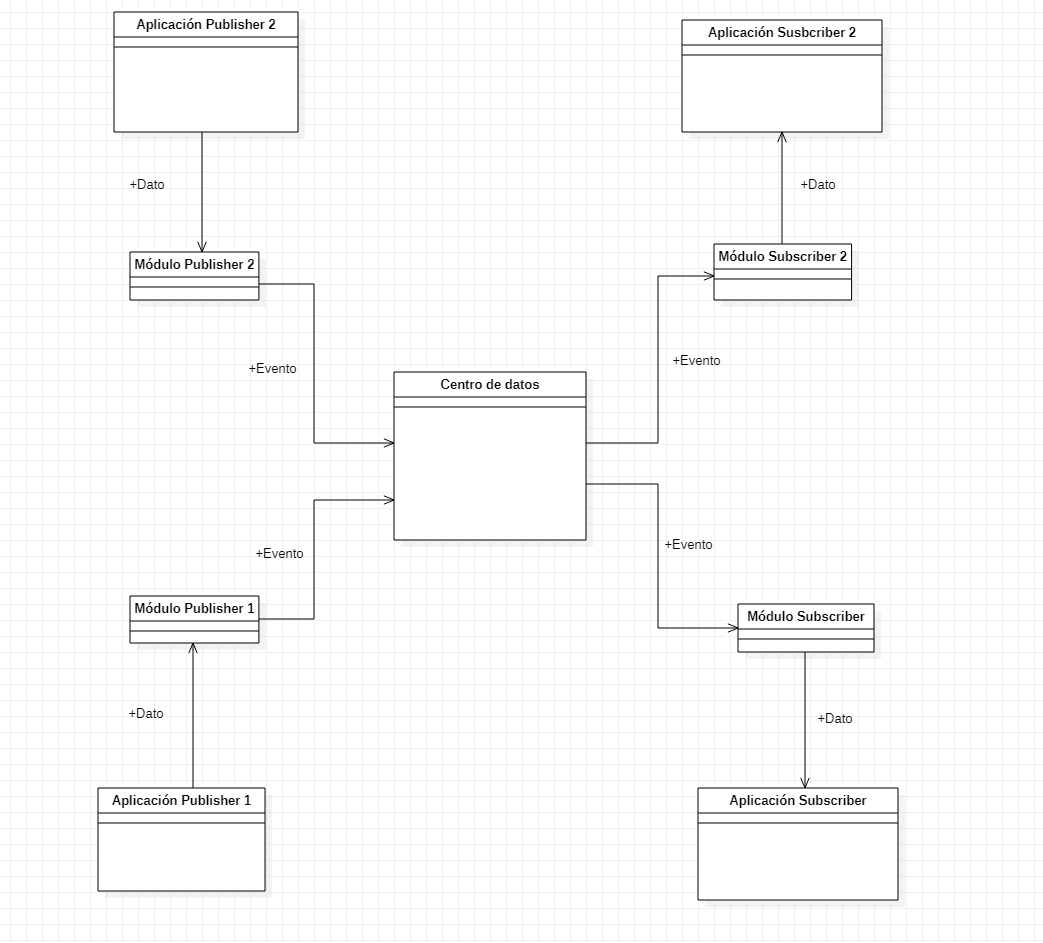
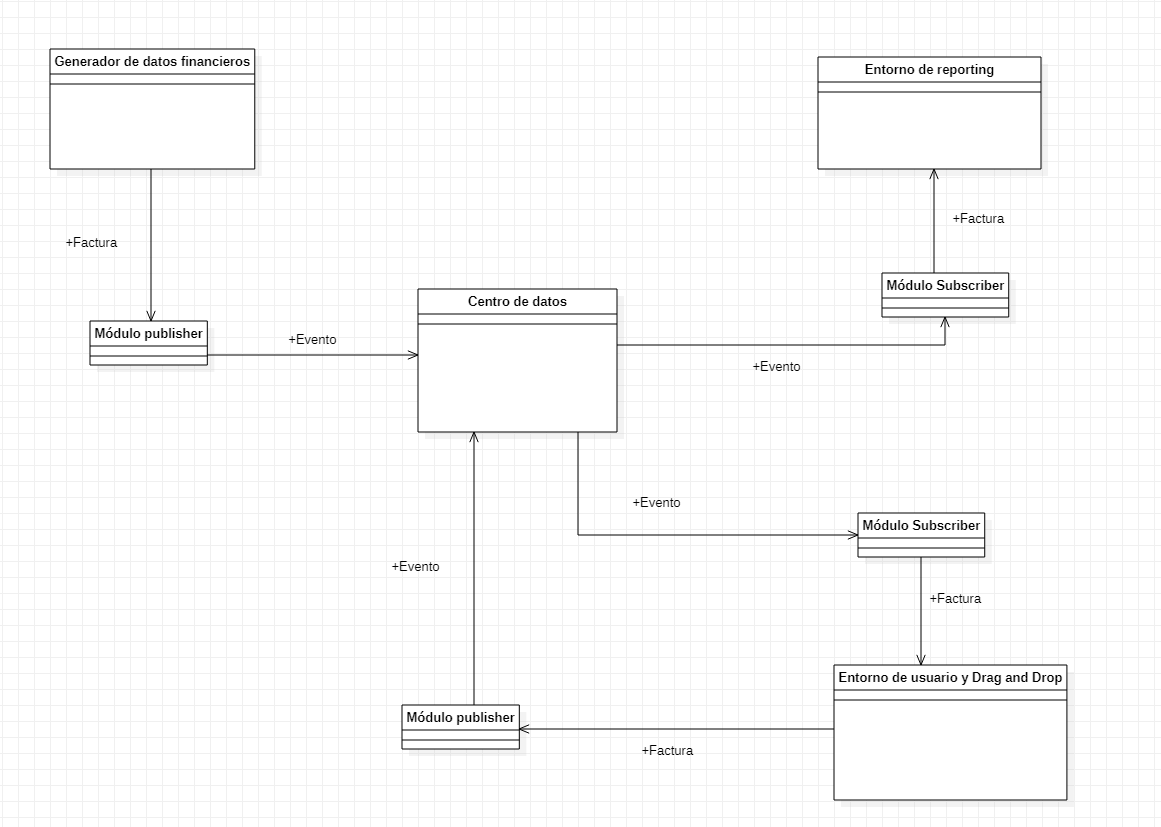


Diagrama arquitectura Publisher/Subscriber

A partir de esta arquitectura, se ha desarrollado Finbook, que está divida en 4 módulos: un centro de datos, un Publisher, un Subscriber y uno mixto (Publisher y Subscriber). A continuación, se explican los distintos módulos:

* El primero es un centro de datos encargado de guardar facturas. Realizado por Raúl Lozano Ponce.
* El segundo módulo se encarga de generar una gran densidad de facturas realistas a través de un entorno empresarial ficticio, generar los archivos de las facturas y publicarlas. Realizado por Juan Alberto Ureña Rodríguez.
* El tercero es encargado de procesar las facturas para calcular las desgravaciones del IVA. Realizado por Juan Kevin Trujillo Rodríguez.
* El último es principalmente un Subscriber que se encarga de calcular el estado financiero, aunque también como Publisher permite subir de facturas ya creadas al centro de datos a través de un “Drag and Drop”. Realizado por Gerardo Santana Franco.

A continuación, se mostrará un diagrama que muestra con más detalle la estructura de nuestro proyecto.



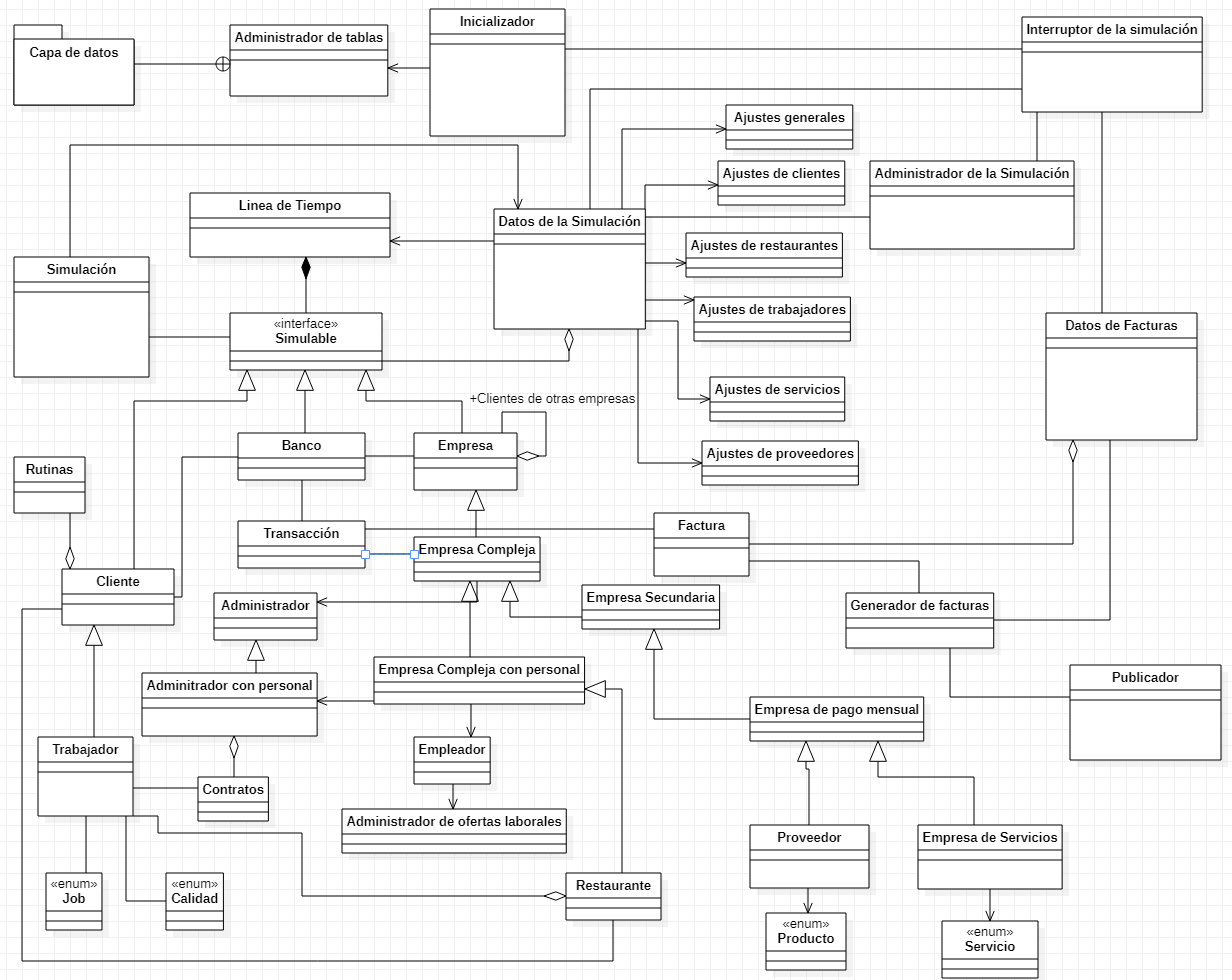
## **5.2 Arquitectura del Módulo desarrollado**

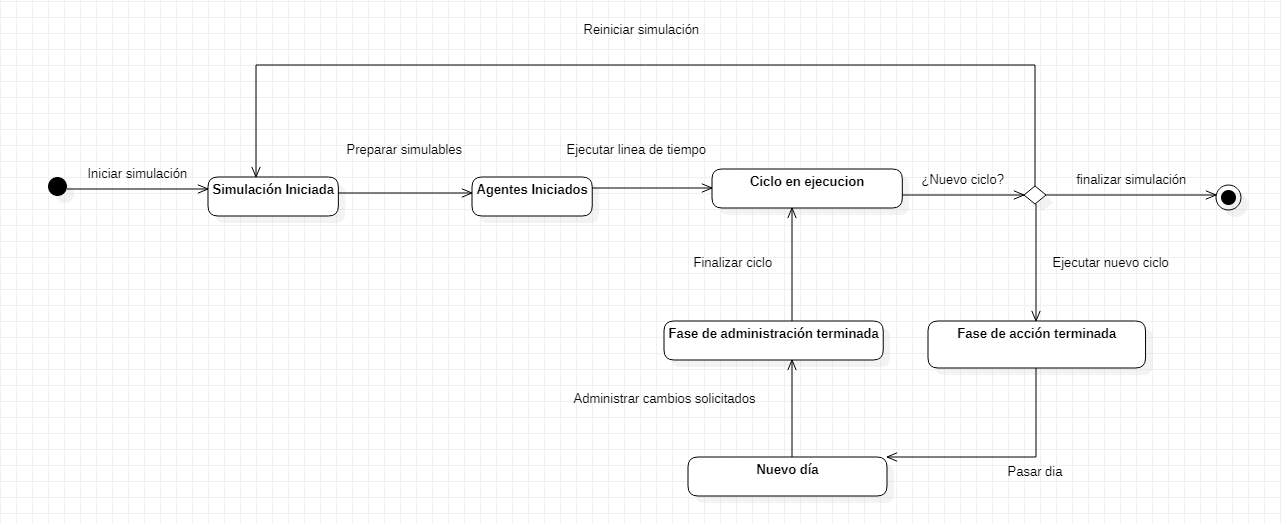
### **5.2.1 Arquitectura Multi-Agente**

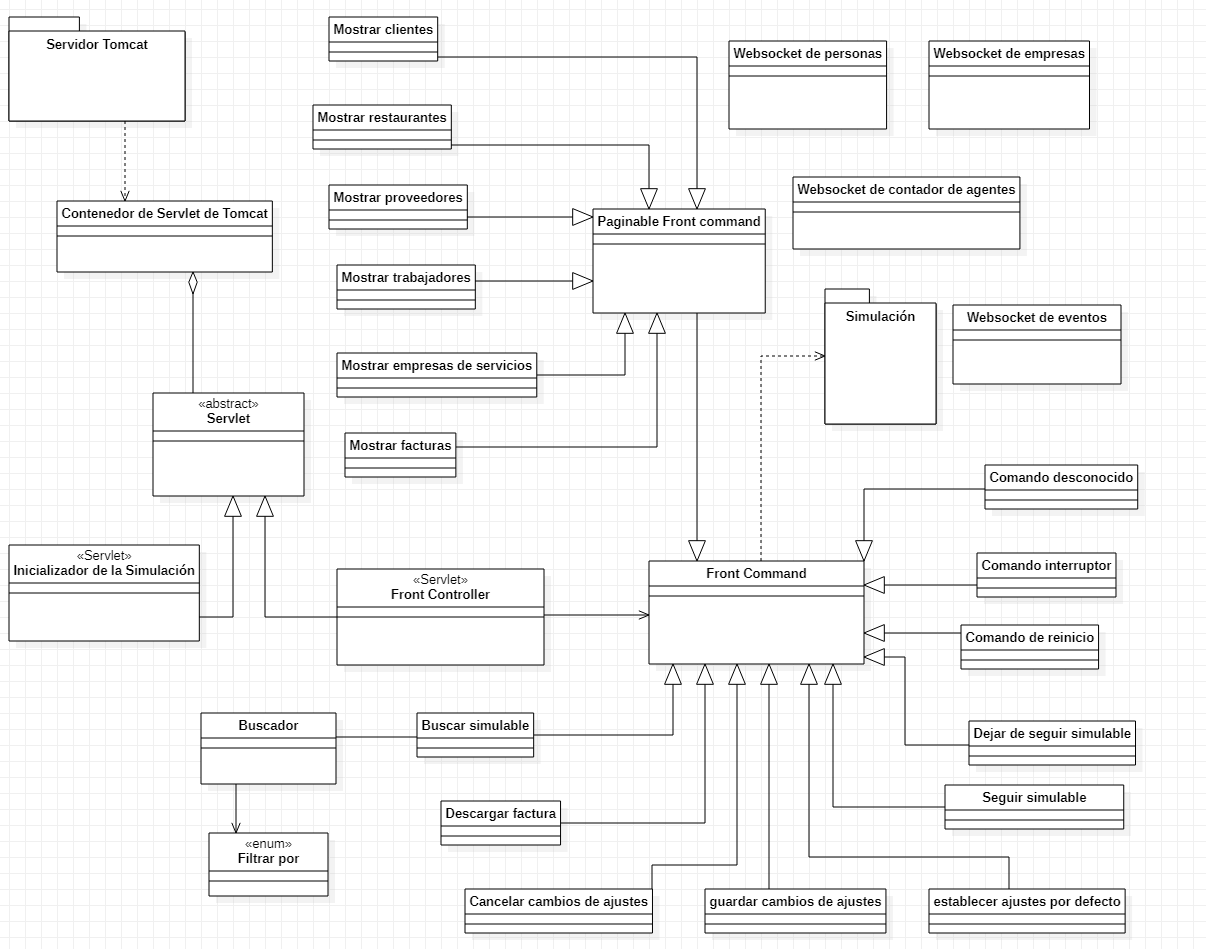
Mi módulo desarrollado ha seguido una arquitectura Multi-Agente, muy común en el desarrollo de simuladores como este caso (15). La arquitectura consiste en la participación de múltiples agentes simultáneos inteligentes que realicen unas acciones en función del rol asociado. Cada ciclo de la simulación, todos los agentes realizan, de forma independiente y asíncrona, todas las acciones que crea oportunas dentro del entorno de la simulación donde se sitúa. Estos agentes tienen cierta inteligencia ya que realizan acciones acordes a los agentes en cuestión en la realidad.

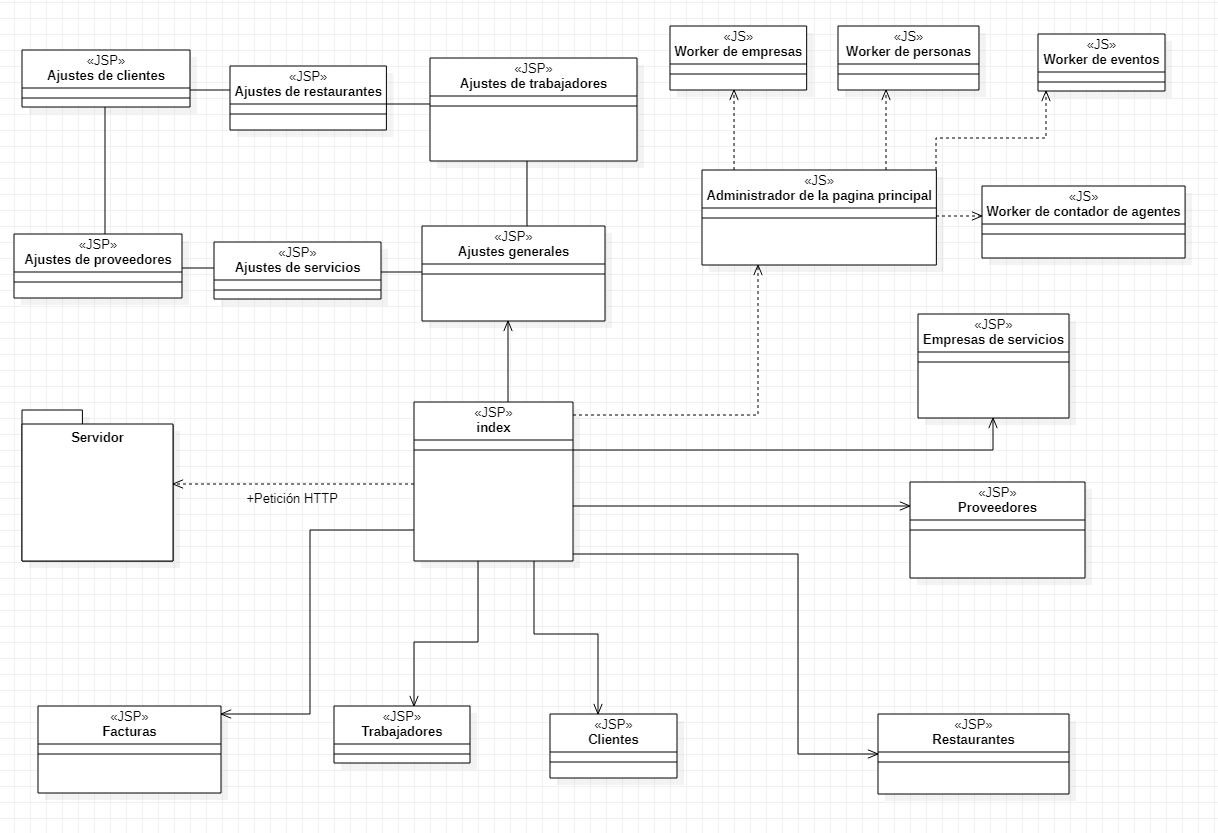
En mi proyecto, el ciclo de la simulación lo dicta la **Línea de Tiempo**, en el cada día de la simulación, cada uno de los agentes participará en el entorno. Esto se explicará durante el apartado del ciclo de vida de la Simulación (7.2.1). Los agentes y sus acciones serán también explicados más adelante, concretamente en la sección de Agentes activos o Simulables (7.2.2).

### **5.2.2 Diseño Modular**









### **5.2.2 Pila Tecnológica**

La pila tecnológica son las dependencias externas de tecnologías que se requieren para la ejecución del proyecto en las que, si faltara alguna de ellas, no se podría ejecutar este proyecto. Las dependencias son las siguientes:

* **Tomcat Server**: servidor necesario para el despliegue del entorno de usuario que ejecutará la simulación. Para más detalles, en herramientas de Software (4.2).
* **DB Browser for SQLite:** aplicación de gestión de base datos necesario para guardar y acceder a los datos de los agentes de la simulación. Para más detalles, en herramientas de Software (4.2).
* **JVM:** Java Virtual Machine es la máquina virtual que permite interpretar interpretar y ejecutar el código binario de java (bytecode Java). (16)

# **6. Competencias Especificas Cubiertas**

Código/s:

- TFG01: Ejercicio original a realizar individualmente y presentar y defender ante un tribunal universitario, consistente en un proyecto en el ámbito de las tecnologías específicas de la Ingeniería en Informática de naturaleza profesional en el que se sinteticen e integren las competencias adquiridas en las enseñanzas.

- CII01: Capacidad para diseñar, desarrollar, seleccionar y evaluar aplicaciones y sistemas informáticos, asegurando su fiabilidad, seguridad y calidad, conforme a principios éticos y a la legislación y normativa vigente.

- CII06: Conocimiento y aplicación de los procedimientos algorítmicos básicos de las tecnologías informáticas para diseñar soluciones a problemas, analizando la idoneidad y complejidad de los algoritmos propuestos.

- CII07: Conocimiento, diseño y utilización de forma eficiente los tipos y estructuras de datos más adecuados a la resolución de un problema.

- CII08: Capacidad para analizar, diseñar, construir y mantener aplicaciones de forma robusta, segura y eficiente, eligiendo el paradigma y los lenguajes de programación más adecuados.

- CII013: Conocimiento y aplicación de las herramientas necesarias para el almacenamiento, procesamiento y acceso a los Sistemas de información, incluidos los basados en web.

- CII014: Conocimiento y aplicación de los principios fundamentales y técnica

- CII015: Conocimiento y aplicación de los principios fundamentales y técnicas básicas de los sistemas inteligentes y su aplicación práctica.

- CII016: Conocimiento y aplicación de los principios, metodologías y ciclos de vida de la ingeniería de software.

- IS03: Capacidad de dar solución a problemas de integración en función de las estrategias, estándares y tecnologías disponibles.

- IS04: Capacidad de identificar y analizar problemas y diseñar, desarrollar, implementar, verificar y documentar soluciones software sobre la base de un conocimiento adecuado de las teorías, modelos y técnicas actuales.

# **7. Generador Sintético de datos**

## **7.1 Introducción**

La explicación de este proyecto realizado junto a sus funcionalidades se dividirá en 2 partes: la simulación y el entorno de usuario que interactuará con la simulación. Cabe destacar que la simulación y el modelo de este se ha desarrollado de tal manera que sea completamente independiente al exterior, es decir, que no tiene ningún tipo de dependencia con el servidor usado el cliente desarrollado, las acciones del usuario o las librerías usadas. No tiene ninguna dependencia exterior con las librerías usadas por lo que, si en el futuro alguna de las librerías o herramientas usadas se quedan obsoletas, no afectará en ningún modo al núcleo del proyecto, a la simulación.

En la primera parte hablaremos del núcleo de la simulación, el modelo del proyecto donde se ejecuta internamente todos los procesos de la simulación y en el que cada uno de los agentes de la simulación realiza sus acciones pertinentes.

En la segunda parte, mostraré la interfaz gráfica del usuario, las funcionalidades que tiene y las opciones en las que el usuario puede influir en la simulación.

Antes de empezar es importante aclarar una palabra que se usará a menudo durante esta explicación, que es el concepto de simulable. Un simulable es un agente de la simulación, pero usamos esta palabra ya que refleja de forma más precisa el concepto, ya que los agentes o simulables van a simular elementos como por ejemplo un cliente, o un restaurante. Estas personas o empresas serán simulables que simularán ser esos elementos durante el transcurso de la simulación. Tomaran decisiones y realizaran acciones acordes al elemento que son, por ejemplo, el cliente ira a comer de vez en cuando en función de su poder adquisitivo a un restaurante que pueda permitirse.

## **7.2 Simulación**

### **7.2.1 Ciclo de Vida de la Simulación**

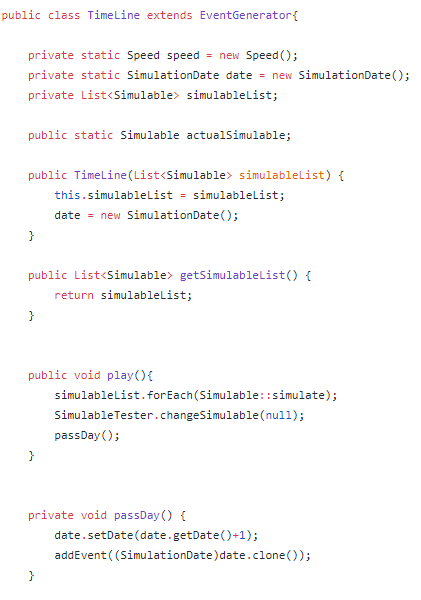
Para explicar esta simulación, es indispensable explicar el proceso principal de la simulación, el ciclo de vida. Este proceso lo lleva a cabo los administradores de la simulación, un conjunto de clases que se explicaran en el punto 7.2.4.

El ciclo es esencialmente un bucle constante que mantiene una fecha que se incrementa un día en cada iteración. Esto tiene como objetivo crear una línea de tiempo en que el ciclo o bucle es el “corazón que bombea” con los días que pasan de forma constante en cada iteración de este.

Dentro de cada simulación se produce cuatro procesos principales, en este orden:

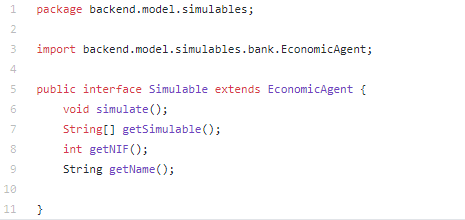
* Se llama a cada uno de los agentes para que realicen la acción que les toque (esto se explicará más adelante con detenimiento).
* Se pasa un día en la simulación.
* Los administradores de la simulación realizan ciertas tareas de mantenimiento y control (7.2.4)
* Se producirá una espera momentánea para que los días tengan una cierta duración de no más de 5 segundos.

En el primero, la línea de tiempo llama a cada uno de los agentes activos de la simulación. Estos agentes o simulables se encuentran guardados en una lista dentro de la línea de tiempo que serán siempre actualizados inmediatamente cada vez que se produzca el alta o la baja de algún simulable. A continuación, se puede ver la clase de la Línea de Tiempo, en la se puede visualizar la lista de los simulables, el método play() que llama a los simulables para que simulen, y el método passDay() que pasa el día.



Clase TimeLine (Línea de Tiempo)

La línea de tiempo no sabe que simulable es realmente, ya que mediante el principio de sustitución de Liskov, la línea de tiempo no conoce las implementaciones que se encuentran detrás y para esta, son simples “Simulables”, es decir una interfaz con el método simulate(), como podemos ver en la foto que se encuentra a continuación. El resto de los métodos son de identificación del simulable que tendrán siempre todos los ellos.



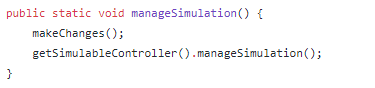
Interfaz Simulable

Los simulables serán llamados uno a uno por la línea de tiempo y realizarán la acción oportuna durante ese día. Estos simulables serán llamados por el método simulate() y en función de las circunstancias realizaran unas acciones u otras. Todos los tipos de simulables que hay y sus acciones serán explicadas durante el apartado 7.2.2.

Cuando terminen todos los elementos se terminará las acciones durante este día y se dará por finalizado este proceso.

Tras la fase de acción de los simulables, se llama al método passDay() que nombramos anteriormente, esta se encarga de actualizar la fecha que tiene la línea de tiempo, además de generar el evento de que ha pasado un día. Este tema de los eventos se explicará posteriormente en el apartado 7.2.6.

A continuación, nos encontramos con la fase de administración. En esta parte, se encarga de administrar la propia simulación y los simulables activos de esta. Debajo se encuentra el método principal de la fase de administración



Método Principal de administración

Este método como se puede observar se divide en dos partes, la primera se encarga de confirmar los cambios solicitados por los simulables y las bajas de simulables, el segundo se encarga de las posible altas en la lista de simulables.

En cuanto a la administración de cambios, esta fase concretamente se produce por el problema que trae consigo ciertos cambios en medio de la fase de acción de los simulables, por ejemplos las bajas de los propios simulables mientras la misma lista está siendo iterada por la línea de tiempo traerá problemas de concurrencia en la ejecución de la simulación. Estos cambios de bajas de simulables se explicarán durante el apartado 7.2.3 y el resto de los cambios se explicarán durante la explicación de los simulables que solicitan esos cambios en el apartado 7.2.2.

En la segunda parte de la fase de administración, nos encontramos la sub-fase que controla las altas de simulables. Se encarga de comprobar si se quiere generar alguno de los simulables y si es afirmativo, se llamará a el módulo concreto de creación e inicialización de simulables y actualiza la lista de simulables de la línea de tiempo. Estas altas de simulables se explicarán con más detenimiento durante el apartado 7.2.3.

### **7.2.2 Agentes Activos o Simulables**

En este apartado veremos todos los tipos de agentes o roles que hay dentro de la simulación, así como sus características sus acciones y su toma de decisiones. Algunos de estos agentes son eminentemente activos (realizan acciones de forma autónoma sin que nadie se lo haya solicitado), y otros son principalmente pasivos (están a la espera de que algún otro agente solicite que realice una acción de forma directa o indirecta. Estos agentes se pueden dividir en 2 tipos principalmente, particulares o personas, y empresas.

Empezaremos por las empresas, desde las menos importantes o con menos acciones con otros agentes, hasta los más importantes o con más acciones.

#### **7.2.2.1 Empresas**

En esta sección expondré todas las empresas existentes en la simulación, los productos o servicios que ofrece y sus acciones. Estas empresas tienen algunos elementos en común:

* Datos de La empresa: Todos tienen unos datos comunes, como el nombre de la empresa, el NIF, la calle, el número de teléfono. Estos datos se generaron a través de generador de datos aleatorios, como explique anteriormente, que genera muchas filas de estos datos. Como excepción, están los restaurantes, que se obtuvieron directamente de Tripadvisor.
* Administración de finanzas: todas las empresas controlan un capital y unos gastos y beneficios ya que se producen compras y ventas de forma constante. Por lo tanto, todas las empresas tendrán un control de esto, que es sencillamente una clase que guarda todos esos datos y realizas toda administración financiera cada vez que se produzca alguna compra o venta. Todas las empresas tienen una instancia propia con datos del capital social inicial, el patrimonio neto, la tesorería, los gastos, las ventas, los beneficios, los datos del mes anterior… etc.
* Clientes de otras empresas: todas las empresas son también clientes de otras que compran productos y/o servicios. Estas empresas se explicarán durante este apartado. Estas empresas se pagarán mensualmente por los productos o servicios aportados. Además, las empresas buscaran los servicios y productos que necesiten para su empresa y buscaran el mejor precio del mercado, cambiando de proveedor si es necesario.
* Pago de impuestos: cada empresa paga unos impuestos mensuales en función de sus beneficios, es decir, un porcentaje de estos que podrá ser cambiado por el usuario en todo momento.
* Pago de hipoteca: cada empresa pagará también mensualmente por el local que usa para su negocio. Este pago dependerá de factores concretos de la empresa.
* Creación y quiebre: las empresas pueden llegar a formarse de forma espontánea si las condiciones son idóneas para que la idea de negocio sea rentable. Además, estas empresas pueden quebrar si se encuentran grandes pérdidas. Esta parte se explicará con más detalle en el apartado de bajas y altas (7.2.3).
* Precio bajo demanda: las empresas suben y bajan los precios de sus productos o servicios en función de la demanda actual. Cuando sus productos o servicios están siendo muy solicitados subirán el precio, en caso contrario, lo bajarán para atraer a más clientes.
* Administrador: todas las empresas tendrán un administrador que les controlará todos los procesos internos como los cambios de precio o de empresa en la que se es cliente.



Clase Administrador

##### Empresas de Servicios

Estas empresas ofrecen algún tipo de servicio a otras empresas, en concreto hay dos tipos de servicio en esta simulación, de transporte o de limpieza. La arquitectura desarrollada permite añadir más tipos de servicios incluso a particulares, pero en esta simulación centrada en la restauración, son los 2 tipos de servicios principales, complementarios a este sector. Ofrecen un servicio a cambio de un pago mensual.

##### Proveedores de Productos

Proveen de productos o materias primas a las empresas para que las usan para crear otros productos o servicios. En este caso, al centrarse en el sector de la restauración, los productos son alimentos principales para los restaurantes como vegetales, carne, pescado, frutas y otros. Cada uno de ellos proveerá de un producto concreto.

La arquitectura permite en todo momento añadir nuevos productos que sean usados para otros motivos y empresas ya que los proveedores son genéricos y cada empresa elige cuales quiere comprar. Al igual que los servicios, estos consisten en un pago mensual a cambio del aprovisionamiento de productos. Además, para el transporte de estos productos, harán servicio de empresas de transporte, explicadas anteriormente.

##### Restaurantes

Es la empresa principal de este sector y, por lo tanto, la que más interacciones tiene dentro de la simulación. Esta empresa compra todos los alimentos necesarios, por lo que necesitará proveedores de todos ellos. Además, también necesitaran servicios de limpieza.

Al ser un restaurante, también necesitará de trabajadores que se encarguen de llevar a cabo el negocio completamente. Estos trabajadores tendrán una labor dependiendo del trabajo, ya sea camarero, cocinero, ayudante de cocina, maître o chef. Los trabajadores no trabajan indefinidamente ya que funcionan con contratos como en la realidad. Esta labor la controlan dos nuevos encargados en la administración, el empleador y el mánager de ofertas.

Al acabarse un contrato, el empleador con cierto tiempo de antelación manda al mánager de ofertas a buscar posibles mejores opciones que el trabajador que acaba contrato, el día que acaba el contrato, el empleador toma una decisión, o quedarse con el trabajador extendiéndole el contrato con una subida de sueldo, o contratar a otro. Esta decisión depende de ciertos factores que el empleador, al igual que la búsqueda por parte del mánager de ofertas:

* Grandes pérdidas: si el restaurante esta con muchas pérdidas económicas, el mánager buscará los trabajadores que acepten el menor sueldo, esto del sueldo aceptado se explicará más detalladamente en la sección del trabajador. El empleador se quedará con el trabajador con menor sueldo de los que encontró el mánager y manda al mánager a obtener el trabajador final, si tienen el mismo sueldo, buscará el trabajador con mayor calidad (esta parte de calidad también se explicará mejor en la sección de los trabajadores). Por último, el mánager comprueba las características de los dos finalistas. Si el trabajador actual cobra menos o igual, se quedará con el actual y renovará el trabajador con un aumento de sueldo. En caso contrario, se despedirá al actual y se contratará al nuevo.
* Estado mediocre: si el restaurante está en un estado aceptable, ni muchas perdidas, ni muchos beneficios, el mánager buscará el trabajador con mejor proporción calidad/sueldo. El proceso es el mismo, pero con esta estrategia.
* Grandes beneficios: si el restaurante está con grandes beneficios, el mánager buscará el trabajador con mejor calidad. El proceso también es el mismo.

Esta toma de decisiones se lleva a cabo con el patrón “Strategy”, <https://refactoring.guru/design-patterns/strategy>.

Los restaurantes mensualmente generan una nómina a todos sus trabajadores con el salario a pagar, llamando al banco para generar la transacción del salario reduciendo su patrimonio, esta parte se explicará con detenimiento en la sección de Otros (7.2.2.2) en la explicación del banco como agente de la simulación.



Clase Empleador



Clase Manager de Ofertas

Los restaurantes aumentarán o disminuirán el precio mínimo y máximo del plato en función de los ingresos, todo ello administrado directamente por el administrador explicado anteriormente, con una extensión, ya que ahora también administran los trabajadores y sus contratos.

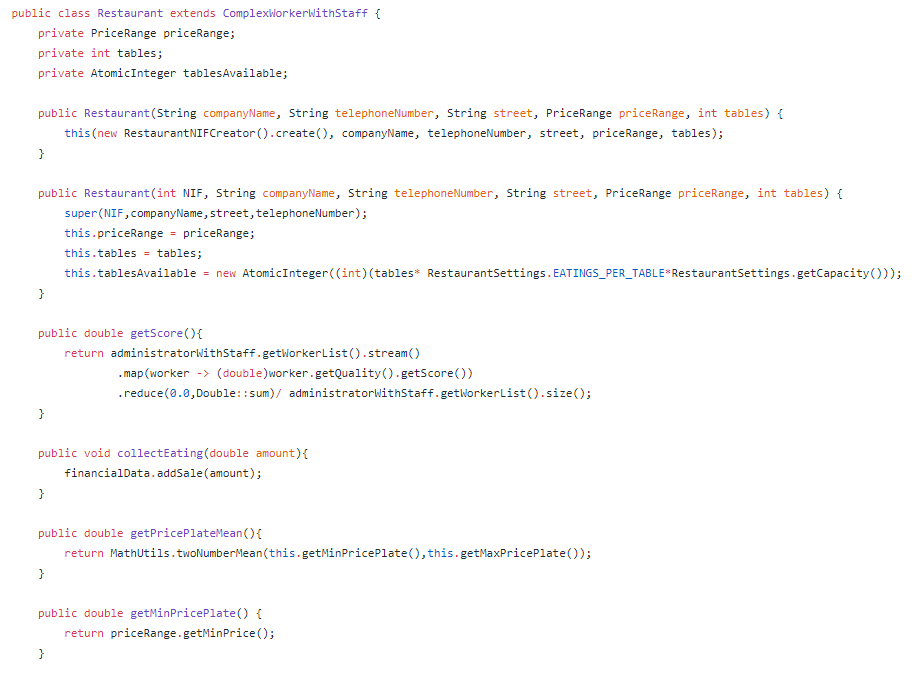


Clase Administrador con trabajadores

Los restaurantes son consumidos por clientes, estos irían a los restaurantes que se pueden permitir y con una frecuencia que depende de su poder adquisitivo. Los restaurantes tienen una calidad dependiendo de la nota media de los trabajadores. Los clientes elegirán entre los mejores restaurantes que están un precio adecuado para ellos. El cliente al elegir restaurante, este solicitará entrar, si hay espacio disponible el restaurante aceptará, si no, lo rechazará

Los contratos tienen una duración variable que podrá especificar el usuario. Los restaurantes necesitan un número concreto de trabajadores de cada trabajo que depende del tamaño del restaurante, y este se mide por el número de mesas. Cuantas más mesas más capacidad de clientes, pero más trabajadores y mayor pago de hipoteca.

Por último, cabe destacar que los restaurantes son restaurantes reales ya que todos sus datos son sacados directamente de la página web del Tripadvisor a través de una lectura de la web o “Web Scrapping”, como expliqué anteriormente.



Clase Restaurante

#### **7.2.2.2 Personas**

##### Clientes

Los clientes son personas corrientes, que se excluye de la simulación su vida diaria, y solo se simula sus interacciones con el restaurante, como comensales. Cada uno tiene sus datos personales, como el NIF, nombre, apellido, fecha de nacimiento, trabajo, salario… etc. Este último, el salario, se genera en la propia simulación cuando se crea el cliente. Este salario se genera a través de una distribución normal que cumple con los datos actuales de salario en España.

Los clientes tienen una rutina, esta es una lista de restaurantes que ir. El tamaño de la lista y el precio de los restaurantes dependerá del poder adquisitivo del cliente. Esto se lleva a cabo a través de una separación en grupos, en función del salario del cliente, estará en un grupo u otro. Los clientes pueden aparecer espontáneamente durante la simulación, y también pueden morir cuando envejecen.

Los clientes dirigen un porcentaje de su sueldo al consumo de restaurantes, este porcentaje fue sacado de estudios al respecto para aumentar el realismo de la simulación. Este porcentaje es el presupuesto que tiene el cliente para el gasto de restauración de forma mensual.

Cada vez que llegue el día de consumo de algún restaurante de la rutina, el cliente primero comprueba si tiene presupuesto como para permitírselo, si no lo tiene, cancela la rutina.

Por último, el cliente puede traer invitados consigo y paga por todos. Este número de invitados varia aleatoriamente. El cliente y sus invitados piden una serie una serie de platos con precio por plato variable en función del precio del restaurante, todo usando distribuciones normales con ciertos límites por lo bajo en el caso de los platos que generan datos muy realistas tras muchas pruebas. Como consecuencia se generan facturas bastante realistas de grupos de clientes que vienen a comer a un restaurante. La parte de facturas se explicará más adelante en la sección de facturas (7.2.5).



Clase Cliente

##### Trabajadores

Los trabajadores son extensiones de la clase Cliente, que además son trabajadores de restaurantes. Hacen las mismas acciones que el cliente salvo que irán a comer a restaurantes si tienen trabajo únicamente. Estos trabajadores también se pueden retirar cuando cumplen una edad concreta (normalmente 65 años, aunque puede ser cambiado por el usuario). Los trabajadores cuando se jubilan hay 2 opciones:

* Si están en paro en el momento de jubilación, obtendrán la pensión con el salario mínimo.
* Si están trabajando, cuando se termine el contrato, el restaurante contratará directamente al mejor trabajador en el mercado según la estrategia actual, y jubilará al trabajador actual cobrando un porcentaje del salario actual como pensión, que podrá ser editado por el usuario.

Los trabajadores tienen una calidad que puede ser muy baja, baja, media, alta y muy alta. Esto afecta directamente a dos factores, a la hora de contratarlos, como hablamos anteriormente, y a la calidad del restaurante, ya que se hace una media de la calidad de todos los trabajadores.

Los trabajadores cuando están en paro tienen un salario deseado a la hora buscar trabajo. Este salario deseado baja conforme pasan los meses sin tener trabajo. Esta acción afecta directamente para ser contratado, porque como expliqué anteriormente, los restaurantes si hay problemas financieros, tendrán en cuenta este salario deseado a la hora de contratar personal. Dependiendo de la situación laboral:

* Mientras tiene trabajo, realizará el trabajo de forma adecuada y contribuirá al consumo de restaurantes como cualquier otro cliente.
* Mientras no tenga trabajo, esperará ofertas de los restaurantes y cuando encuentra alguna que le satisfaga en cuanto al salario deseado, aceptará. Si tiene varias, elegirá la mejor opción. Además, al no tener trabajo no consumirá ningún restaurante.

Estas ofertas funcionan de una manera muy simple. Los mánager de ofertas mandan una batería de ofertas a muchos trabajadores con el criterio de la estrategia aceptada. Esta oferta tiene 2 firmas, la del restaurante y la del trabajador. Cuando el trabajador acepta esta oferta, llama al método adecuado para aceptar esa oferta. El mánager cuando es la hora de obtener la lista de candidatos, este obtiene de esa lista de ofertas solo las ofertas que estén aceptadas por los trabajadores.



Clase Oferta de Trabajo



Clase Trabajador

#### **7.2.2.3 Otros**

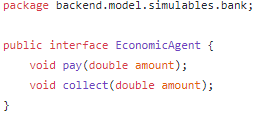
##### Banco

Es una entidad central que controla el flujo de capital entre las distintas entidades. Cada vez que se produce algún tipo de compra o venta, esta entidad única en la simulación es la llamada para realizar la transacción, como elemento central. Todo los simulables conocen al banco y cada vez que haga falta alguna transacción en alguna compra o venta, llamaran al banco para que la realice.



Clase Banco

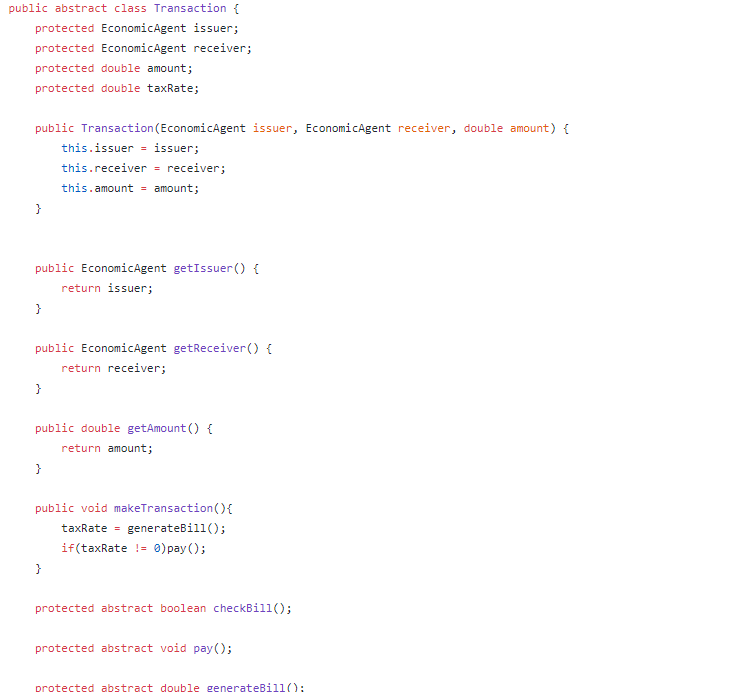
Al contrario, el banco no sabe quiénes son, estos están envueltos en una interfaz genérica que controla los cambios que se producen en la transacción en cada parte. Esta interfaz se llama Agente Económico (EconomicAgent) y tienes dos métodos a implementar por cada simulable, pay() y collect(). La propia interfaz simulable que vimos anteriormente, la cual todos los agentes tienen que implementar, también extiende de la interfaz EconomicAgent, por lo que todos los agentes de la simulación siempre serán agentes económicos que implementen un método de pago y de cobro.



Interfaz Agente Económico

Las empresas implementan esta interfaz, incrementando o disminuyendo el patrimonio neto. Los clientes la implementan simplemente reduciendo o aumentado el presupuesto para este mes.

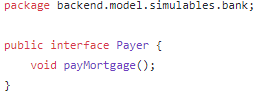
En cuanto a las transacciones, son simplemente instancias que genera el simulable para emitir la factura. El simulable manda la transacción al banco para que ejecute y hacer los pagos. Estas transacciones son las encargadas de generar la factura internamente llamando al generador de facturas que, como es habitual, se explicará en su sección correspondiente (7.2.5). Por cada tipo de factura, hay un tipo de transacción. Cuando se produce la transacción, se añade el porcentaje en impuestos al en el pago al pagador.



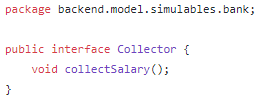
Clase Abstracta Transacción

El banco es también un agente activo ya que este, cuando es el último día del mes se encarga de, mensualmente, hacer pagar la hipoteca del local de todas las empresas.

Por último, nos encontramos con dos interfaces que implementan algunos la primera, y otros la segunda. Estas son las interfaces de pagadores (payers) y cobradores (collectors). Los payers son siempre empresas, que tienen que pagar mensualmente unos impuestos al estado, este impuesto, como explicamos anteriormente, es un porcentaje del beneficio de la empresa que puede ser cambiado por el usuario. Los collectors por su parte, son los clientes y trabajadores que mensualmente cobran el sueldo del banco. Estas juntos a las hipotecas son las tres acciones activas del banco.



Clase Pagador



Clase Cobrador

### **7.2.3 Bajas y Altas de Simulables**

Los simulables, como explicamos anteriormente, pueden entrar y salir de la simulación. Estas entradas y salidas dependen de muchos factores. Los encargados de controlar estas entradas y salidas son los administradores de la simulación que, en cada iteración del ciclo de vida durante la fase de administración, se controla si alguien quiere salir, o si se cumplen los factores para entrar de algún simulable y se inicializa el simulable.

En cuanto a las salidas, las personas a partir de cierta edad (75 años) todos los días hay una ínfima probabilidad de morir. Esta probabilidad está lo suficientemente ajustada para que las personas tengan una esperanza de vida que siga con el estándar de este país (84 años de media).

Las empresas, por otro lado, quebrarán si las finanzas no son favorables y los beneficios son negativos e inferiores a una cantidad, que puede ser cambiada por el usuario.

En cuanto a las entradas, estos simulables se generan si la situación es propicia y rentable. Si se propicia y se cumple los factores para alguno de los simulables, se inicializará un simulable de ese tipo y se añadirá a la simulación. Estas entradas dependen de una formula, que es una probabilidad que cuanto mayor sea esta, más probabilidades que se genere tal simulable. Esas probabilidades son todas proporciones que reflejen realista y lógicamente la realidad. Estas dependen de cada simulable:

* Cliente: Cuanta menor proporción de clientes por cada restaurante haya, más posibilidad de que se generen y viceversa.
* Restaurante: Es la proporción contraria al cliente, cuantos más clientes haya por cada restaurante, más probable.
* Trabajador: cuanto menor sea el porcentaje de paro, mayor la probabilidad de que se genere un trabajador.
* Proveedor de productos: cuanto menor proporción haya de proveedores por cada restaurante, mayor la probabilidad de que se genere un proveedor.
* Empresa de servicios: cuanto menor proporción sea la proporción de las empresas de servicios en el número de empresas, mayor la probabilidad de que se genere una empresa de servicios.

La manera en la que se generan estos simulables se explicará durante el apartado de Preparación de Simulables (7.2.8). Este mecanismo de generación de simulables es bastante realista y controlado para que el sistema sea estable y cumpla con lo visto en la realidad.

### **7.2.4 Administración**

#### **7.2.4.1 Control Central**

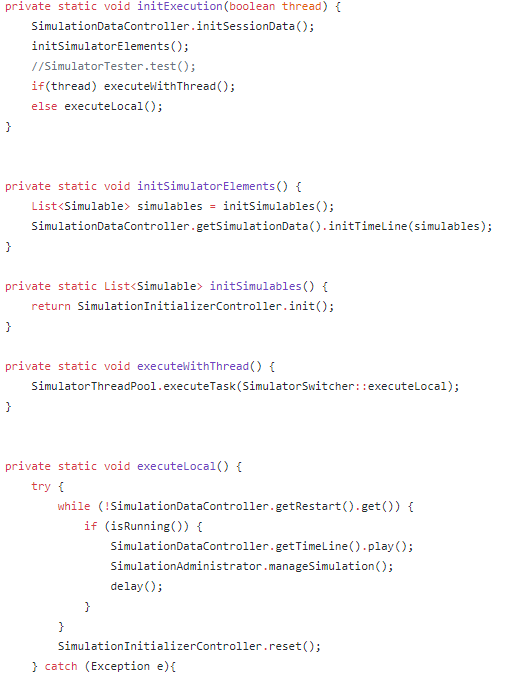
##### Interruptor del Simulador

Es el encargado de controlar la ejecución de la simulación, la clase que actúa como interruptor de la simulación, que llamas para parar y encender la simulación. También, permite reiniciar la simulación.



Clase SimulatorSwitcher (Interruptor del Simulador)

Cuando se llama a ejecutar por primera vez, se envía un boolean (true o false) que indica si se ejecuta en un hilo a parte o no. La ejecución se puede realizar en local usando simplemente la consola para mostrar los eventos (apartado 7.2.6), o en la web con un servidor de Tomcat. La primera opción, al ser en local y sin ningún tipo de interacción, se ejecuta sin hilo. La segunda al tener incorporado una interfaz de interacción con el usuario, se ejecuta en un hilo aparte.



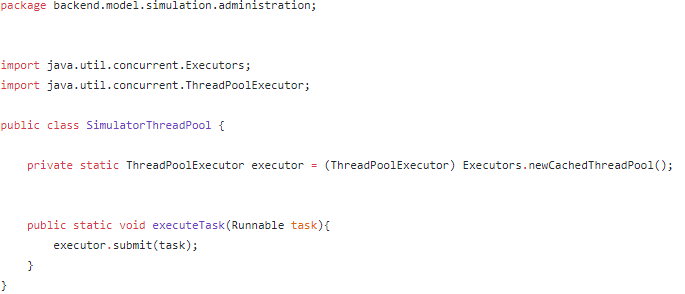
Inicialización de la Simulación

La administración del hilo se lleva acabo usando un ThreadPoolExecutor de la librería Concurrent (11). Esta te permite crear un “pool” de hilos que tu puede ser para tu proyecto. El objeto lo puedes usar mandándole tareas que puedan ser ejecutadas y esta te ejecutará la tarea en un hilo aparte mientras haya espacio. Cuando lo creas puedes ponerle un tamaño máximo de hilos o un tamaño variable.

En mi proyecto cree una clase que envolvía ese objeto y se usa como clase de acceso al ThreadPoolExecutor. El interruptor del simulador llama a esta clase mandando una tarea (Runnable Task) a la clase.



Llamada de la clase mandando la tarea.



Clase SimulatorThreadPool

Esto permite una arquitectura fácilmente adaptable a la ejecución de varias simulaciones simultáneas sin alterar el proyecto.

##### Simulación

Esta clase es la clase principal de servicio a los simulables. Provee de todo tipo de métodos para obtener todo tipo de datos variados según las necesidades de los simulables. Por ejemplo, permite obtener la lista de los trabajadores sin empleo o con empleo, la lista de proveedores de un producto, el número de simulables de cada tipo… etc.

Estos datos tendrán acceso todos los simulables como los restaurantes a la hora de buscar trabajadores para su negocio, o empresas de servicios para la limpieza, o los clientes para buscar restaurantes al que ir a comer.

En definitiva, provee de una serie de métodos “Getters”, que los simulables podrán consumir en todo momento. Además, tendrán acceso a las estrategias genéricas para la selección de ofertas por parte de los trabajadores o la búsqueda de rutinas, por parte de los clientes. Por último, ofrece a la web una interfaz de acceso a los simulables con paginación para poder mostrarlos correctamente al usuario final.

  
Clase Simulación

##### Administrador Principal de la Simulación

Es la interfaz principal de comunicación entre administración y los simulables en cuestión de solicitar cambios por parte de los simulables y es el director principal de la fase de administración en el ciclo de vida, el que dirige todos los pasos de esta fase. Los simulables tienen una gran variedad de acciones posibles, pero muchos de estas acciones pueden causar problemas de concurrencia como por ejemplo la salida de estos en medio de la iteración de la lista de la fase de acción de los simulables o como cambios en el personal o de empresas clientes en una empresa.

Por lo tanto, todos estos cambios de personal, de relaciones de empresas o de salida y entrada de simulables, se solicitan durante la fase de acción de los simulables y se confirmas los cambios (“commit changes”) durante la fase de administración del ciclo de vida. Cuando un agente quiere realizar un cambio importante, como los explicados anteriormente, solicita directamente al administrador principal de la simulación. Esta manda al **Controlador de Simulables** para que añade esta solicitud, para que durante la fase de administración este controlador realice este cambio, esta parte se explicará durante la administración de simulables (apartado 7.2.4.4) donde se explicará las funciones del Controlador de Simulables y otros controladores.



Clase SimulationAdministrator (Administrador de la Simulación)

#### **7.2.4.2 Control de Datos**

##### Datos de Simulación

Es la clase encargada de guardar todos los datos de simulación en ejecución: los simulables activos, la línea de tiempo, los datos de ejecución del interruptor, los datos de ajustes (apartado 7.2.4.5), el controlador de simulables (apartado 7.2.4.4) y la lista con los simulables que se están siguiendo. Esta última se explicará durante el entorno de usuario ya que es una funcionalidad del usuario.

El resto de las clases de administración como la clase Simulation o la clase SimulationAdministrator usan directamente estos datos a través del intermediario SimulationDataController (Controlador de datos de la simulación), que se explicará en el siguiente subapartado.

##### Controlador de Datos de la Simulación

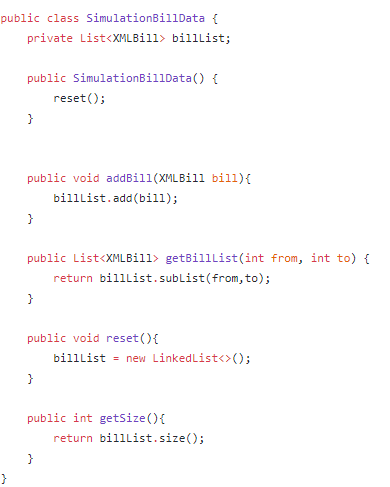
Esta clase es la clase de acceso directo a los datos de Simulación, esta clase tiene como objetivo dividir bien la estructura para poder cambiar fácilmente la simulación con el propósito de ejecutar múltiples simulaciones simultaneas, y esta clase sea la encargada de coordinar a cuál de los datos de las simulaciones acceder en todo momento. A pesar de ello, esta clase sigue siendo muy útil para tratar el SimulationData como una instancia de los datos de la simulación y el SimulationDataController sea la clase estática de fácil acceso a los datos sin tener que instanciar ninguna clase.



Clase SimulationDataController

##### Datos de Facturas de la Simulación

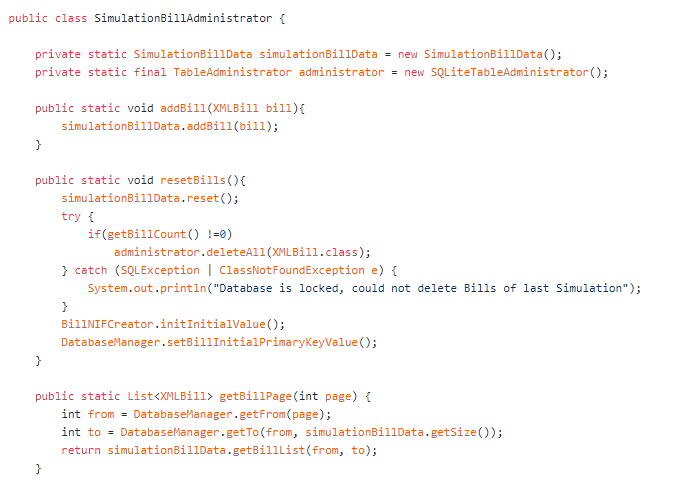
Contiene la lista de todas las facturas generadas durante la simulación actual. Permite reiniciar la lista al reinicio de la simulación.



Clase SimulationBillData (Datos de Facturas de la Simulación)

##### Controlador de Facturas de la Simulación

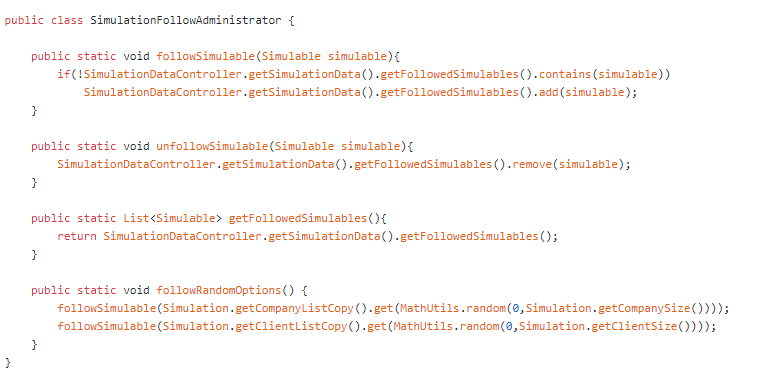
Al igual que el controlador de datos de simulación, gestiona el acceso a los datos, pero en este caso a los datos de las facturas generadas. Tiene el mismo objetivo que en el caso del controlador de datos, usar la clase que guarda los datos como una instancia que se genera en cada simulación y este controlador la clase estática que facilita el acceso a estos. Además, permite como en el caso anterior generar una arquitectura como simulaciones simultáneas.



Clase SimulationBillAdministrator (Controlador de Facturas de la Simulación)

##### Controlador de Seguidos de la Simulación

Esta clase se encarga de controlar los simulables que sigue el usuario de la simulación. Este concepto de seguir la simulación se explicará durante la explicación del entorno de usuario. Además, se encarga de añadir unos simulables aleatorios al inicio de la simulación (también se explicará en dicho apartado).



Clase SimulationFollowAdministrator (Controlador de Seguidos de la Simulación)

#### **7.2.4.3** **Inicializador de Simulables**

##### Inicializador de Simulables

Es el miembro encargado de inicializar los simulables de la aplicación. Este solo controla la inicialización, pasándole el número de simulables de un tipo que inicializar, para luego realizar las funciones necesarias para devolver los simulables requeridos. Permite tanto inicializar una lista como crear uno solo.

Para crear los simulables, primero tiene que acceder a estos que se encuentran guardados en la base de datos, para acceder a ellos usará el paquete desarrollado personalmente para el acceso a la base datos. Para ello usa la clase TableAdministrator que envuelve perfectamente toda la implementación de lectura de datos de las tablas que se encuentran en la base de datos. Esta sección se explicará con más detenimiento durante el apartado 7.2.7 (Capa de Datos).

Además de crearlos, es el encargado de organizar la preparación de los simulables para su entrada a la simulación. Dependiendo del simulable a preparar llamará a unos preparadores u otros que se encargan de darle los elementos necesarios al simulable para que puede empezar a participar en la simulación. Esta parte de la preparación se explicará con más detalle durante el apartado de Preparación de Simulables (7.2.8).

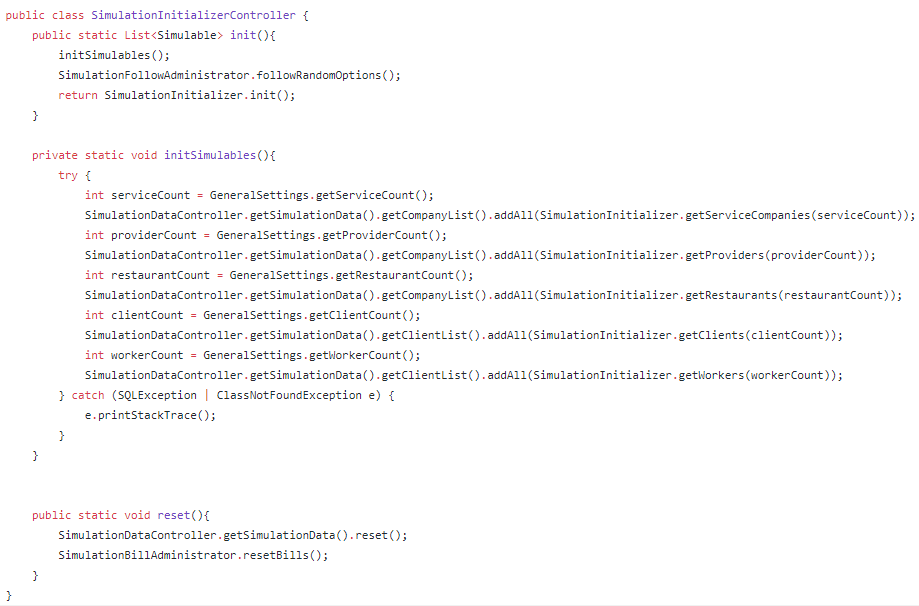
Durante la simulación, se irá avanzando en la lista de los simulables que se van leyendo y desapareciendo otros. Cuando se llega al límite de datos de la base de datos, vuelve automáticamente a empezar desde el principio de la tabla de la base de datos.



Clase SimulationInitializer (Inicializador de Simulables)

##### Controlador de Inicialización de Simulables

Es el encargado de inicializar los simulables iniciales de la simulación, es decir los agentes que se generan al principio de forma automática con el que poder empezar a simular correctamente. Para ello usará la clase **Inicializador de Simulables** explicada en el apartado anterior. Además, se encarga de reiniciar los datos de la simulación (SimulationData) y los datos de facturas de la simulación (SimulationBillData) cuando se solicite reiniciar la simulación por parte del usuario.



Clase SimulationInitializerController (Controlador de Inicialización de Simulables)

#### **7.2.4.4 Administración de Simulables**

##### Committer de la Simulación

Es el encargado de perpetrar (confirmar y realizar) los cambios solicitados por los simulables. Esta clase es usada directamente por el resto de los administradores de simulables, ya que cada uno tiene su propia instancia de esta clase para requerir sus servicios cada vez que sea necesario para realizar un “commit” o confirmación de algún cambio.



Clase SimulationCommitter (Committer de la Simulación)

##### Controlador de Entrada/Salida de la Simulación

Es administrador principal de las entradas y salidas de la simulación, y es encargado administrar las solicitudes de entrada y de salida de la simulación. Para ello, como explicamos anteriormente, hará uso de un **Committer de la Simulación**, para perpetrar estas entradas y salidas.

En cada ciclo de la Simulación, durante la fase de administración, este administrador será llamado por el **Controlador de Simulables** (que se explicará a continuación). Este se encargará de añadir, si es propicio y por cada tipo de simulable, un nuevo simulable de este tipo, como ya explicamos anteriormente durante el apartado de Bajas y Altas de simulables (apartado 7.2.3). Por cada simulable existe una probabilidad que varía según unos factores que afectan directamente al agente (para más información, apartado 7.2.3). Si esta probabilidad se cumple, entonces este controlador añadirá un nuevo agente de este tipo.

Además, durante la realización de los cambios solicitados por los simulables (que se explicará en la explicación del siguiente y último administrador), los cambios solicitados de Salida serán realizados y administrados directamente por este, mediante la ayuda del **Committer de la Simulación**.

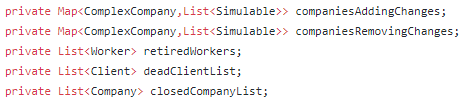


Clase Simulation I/O Controller (Controlador de Entrada/Salida de la Simulación)

##### Controlador de Simulables

Es el director principal de la fase de administración, es llamado por el administrador principal de la simulación para organizar y realizar los cambios solicitados por los simulables y realizar las altas necesitadas.

Como explicamos en la sección del **Administrador Principal de la Simulación**, cada vez que un simulable quiere realizar un cambio, este lo solicita al administrador principal, para el cual manda al controlador de simulables a registrar la solicitud. Esta solicitud se registra mediante una serie de lista que administra y guarda el **Controlador de Simulables** que guarda distintos tipos de cambios, como la lista de empresas que quieren quebrar hoy, o la lista de cambios internos de alguna empresa, ya sea cambios en el personal, productos o servicios.



Listas de cambios solicitados

En cada ciclo de simulación, este controlador se encarga de realizar 2 acciones: realizar los cambios solicitados por los simulables, y llamar al **Controlador de Entrada/Salida** para añadir los simulables necesarios.

Durante la primera acción lee cada uno de los cambios de las distintas listas y realiza los cambios mediante el uso del **Committer de la Simulación**. En caso de las solicitudes de salida, llama al **Controlador de Entrada/Salida** para realizar la salida. Tras realizar todos los cambios, vacía todas las listas de solicitudes de cambios.

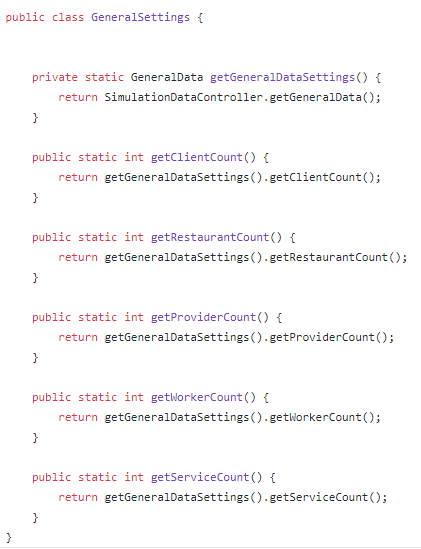


Clase SimulableController (Controlador de Simulables)

#### **7.2.4.5 Ajustes**

Todos los datos que afectan directamente a la simulación, ya que son accedidos durante la simulación han sido implementado mediante una serie de ajustes con un valor por defecto, que puede ser cambiados por el usuario en el apartado de ajustes (que se explicará durante en el apartado del Entorno de Usuario de la Simulación, 7.3). Hay varios tipos de ajustes:

* Ajustes Generales: Controla el número de simulables iniciales de cada tipo con los que empezará la simulación.



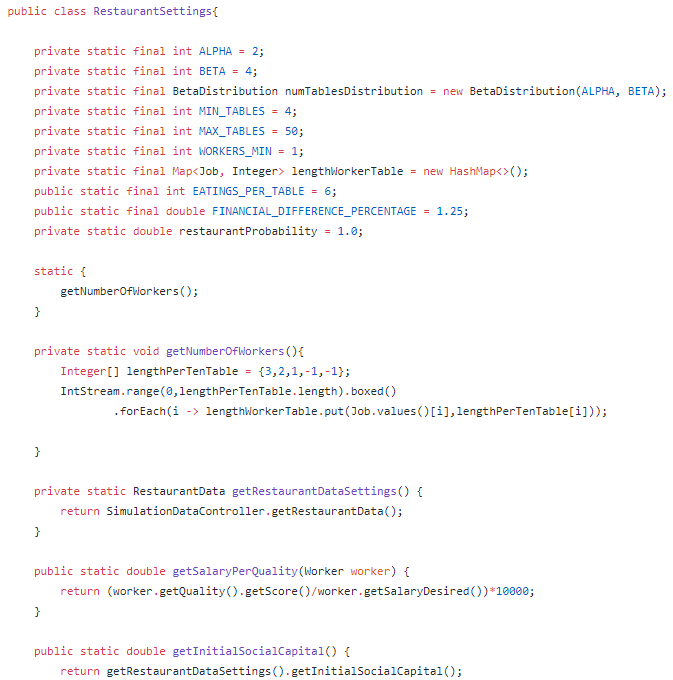
Clase GeneralSettings (Ajustes Generales)

* Ajustes de Cliente: los ajustes que afectan directamente a los agentes que tengan el papel de clientes de la simulación, ya sea para obtener el sueldo que tendrá, la edad a partir de la cual, puede morir...



Clase ClientSettings (Ajustes del Cliente)

* Ajustes de Restaurantes: influyen directamente en el funcionamiento de los restaurantes, ya sea especificando el salario inicial de los trabajadores al principio de la simulación, o el número de mesas de los restaurantes… etc.



Clase RestaurantSettings (Ajustes de Restaurantes)

* Ajustes de Proveedores: ajustes para los datos importantes de los proveedores, como el precio inicial de los productos, o la probabilidad de que el producto salga defectuoso… etc.



Clase ProviderSettings (Ajustes de Proveedores)

* Ajustes de Servicios: son los ajustes relacionados con los servicios y las empresas de servicios, por ejemplo, el precio inicial de los servicios o el capital inicial de estas empresas. Este último dato lo tiene todas las empresas.



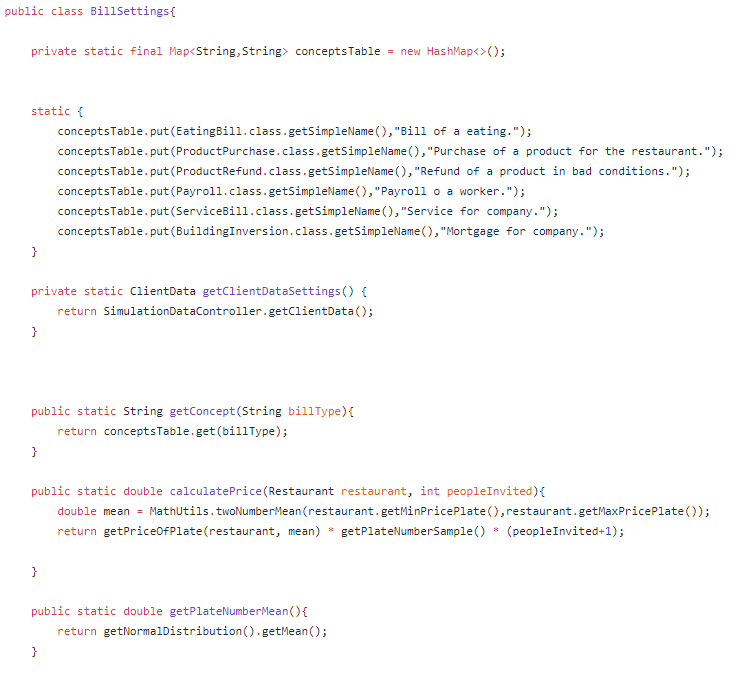
Clase ServiceSettings (Ajustes de Servicios)

* Ajustes de Trabajadores: ajustes que afectan a los trabajadores de empresas. Por ejemplo, la edad de jubilación, o el porcentaje del sueldo que se llevan al jubilarse.



Clase WorkerSettings (Ajustes de Trabajadores)

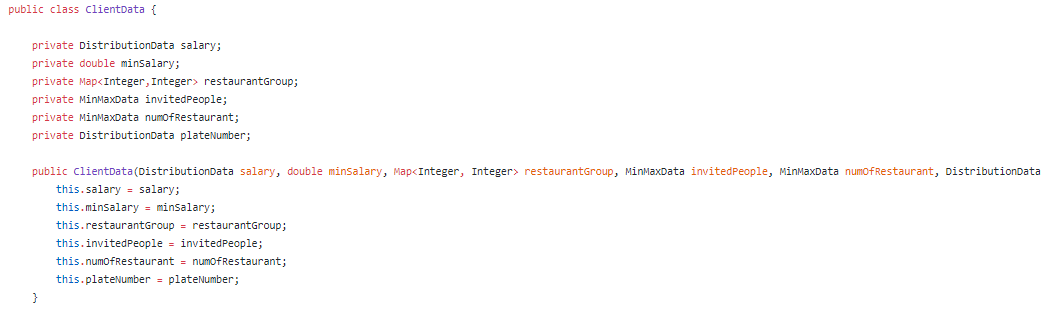
* Ajustes de Facturas: ajustes de las facturas que se generan en la simulación. Estos ajustes son muy poco editables por el usuario, ya que solo pueden ser editados ajustes afectados por ajustes de clientes en el caso de las facturas de comidas, como el número de platos que se pedirán a través de una distribución normal. El resto son ajustes de las facturas que no es beneficioso que puedan ser editados por el usuario, porque son datos muy complejos como el cálculo del precio de las comidas, o los conceptos que se pondrán en las facturas que es indiferente para el usuario.



Clase BillSettings (Ajustes de Facturas)

Estos ajustes tienen un objeto que tiene todos los datos que serán usados para poder proporcionar los servicios dados en su clase. Las clases de datos de ajustes son las siguientes:

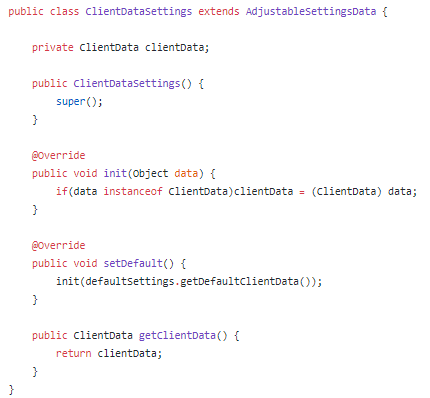
* Datos Generales
* Datos de Clientes
* Datos de Restaurantes
* Datos de Proveedores
* Datos de Servicios
* Datos de Trabajadores



Ejemplo de Clase de datos de Ajustes, Clase ClientData (Datos de Cliente)

Como habíamos dicho anteriormente, las facturas no tienen muchas personalizaciones, y las disponibles provienen realmente de los datos de clientes, por eso no hay datos de facturas para los ajustes.

Realmente, las clases de ajustes no tienen acceso a estos datos, sino a una clase intermedia que controla los cambios de usuario en los ajustes. Estas clases envuelven la clase de datos correspondiente con una interfaz que permite actualizar la clase de datos fácilmente con otra nueva, además de proporcionar la función de devolver los datos de ajustes a su versión por defecto.



Ejemplo de clase con actualización de datos de ajuste, Clase ClientDataSettings (ajustes de datos de cliente)

Estas clases de actualización de ajustes comparten una instancia estática de DefaultSettings (Ajustes por Defecto), es decir, que es única la instancia para todos, y que tiene todos los ajustes por defecto de la simulación. Esta clase al principio de la simulación accede a los datos por defecto que se encuentran guardados en la base de datos, esto se explicará más detalle durante el apartado de la base de datos en la sección de la capa de datos (7.2.7.1).



Clase DefaultSettings (Ajustes por defecto)

Los cambios de ajustes los administra el SettingsBuilder (Constructor de Ajustes). Esta clase provee de una interfaz sencilla para actualizar los ajustes o ponerlos por defecto. La clase recibirá, en el caso de actualización los datos de los distintos ajustes, y se encargará de actualizar cada una de las clases que envuelven y permiten actualizar los datos de estos ajustes.

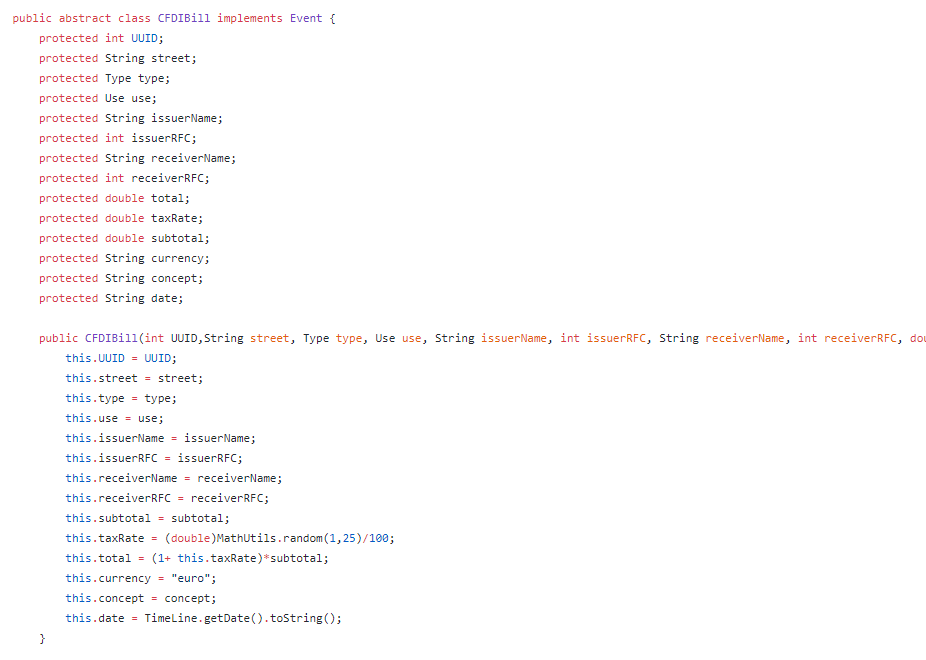


Clase SettingsBuilder (Constructor de Ajustes)

### **7.2.5 Facturas**

Uno de los objetivos principales de la simulación es la generación de facturas, para luego poder ser publicadas en el Datahub del otro módulo principal, y a su vez, el entorno de Usuario pueda usarlas para mostrarlo las funcionalidades del proyecto. Todas las facturas generan eventos cuando se crean, este apartado de eventos se explicará durante la sección 7.2.6.

Las facturas las genera un generador de facturas desarrollo internamente. Todas las facturas proceden una misma clase (CFDIBill) que tiene todos los campos genéricos de las facturas, en el que cada tipo de factura rellena los campos acordes al tipo de factura, estos datos proceden de elementos genéricos que se reciben al generar la factura. Por ejemplo, las facturas de comidas que se generan cuando un cliente consume un restaurante, se generan a partir de tres datos: el restaurante que se consume, el cliente que lo consume, y el coste final de la comida. A partir de estos datos, la clase sabe cómo rellenar los distintos datos de la factura.



Clase CFDIBill (Factura CFDI)

Los campos de las facturas que se generan en esta simulación son los siguientes:

* UUID
* Localización
* Tipo
* Uso
* Nombre del emisor
* RFC del emisor
* Nombre del receptor
* RFC del receptor
* Subtotal
* Tasa de impuestos
* Total
* Moneda
* Concepto
* Fecha de Emisión

Al ser las facturas genéricas, el generador de facturas no conoce que tipo de factura está generando, ya que obtiene una CFDIBill con los campos siempre rellenados, que luego esta simplemente usa para rellenar la factura en formato XML final. Las facturas XML se guardan en la carpeta XMLFiles. Por cada tipo de factura hay una carpeta dentro de la carpeta anterior.



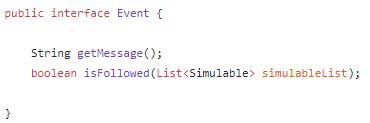
Clase CFDIBillGenerator (Generador de Facturas CFDI)

Los distintos tipos de facturas que se generan son las siguientes:

* Facturas de comidas: son facturas de compras que generan los clientes al consumir comida en los restaurantes.
* Facturas de Productos: son facturas de productos que se compran a proveedores
* Devolución de Productos: son facturas que se generan cuando se produce la devolución del algún producto.
* Nómina: son facturas que se generan cuando se paga el sueldo a los trabajadores mensualmente.
* Servicio: facturas de servicios ofrecidos a un particular o empresa.
* Inversión de edificio o hipoteca: inversiones que se producen al pagar por un local que usar para realizar el negocio.

### **7.2.6 Eventos**

Los **Eventos** son mensajes que se generan cuando se producen ciertas acciones dentro de la simulación que son provocados por uno o más simulables de la simulación. Estos mensajes se muestran al usuario como modo de informar o reportar al usuario de los acontecimientos que han ocurrido en la simulación. Esta es la manera de informar que he usado para este proyecto ya que es un proceso eminentemente cerrado e interno, quedando fuera debido a falta relevancia, la interfaz gráfica que muestre a estos elementos. Tiene un método para obtener el mensaje y otro para saber si ese evento está relacionado con algún simulable seguido, esto último se explicará con más detalle durante el apartado de simulables seguidos dentro del apartado del Entorno de Usuario (7.3).

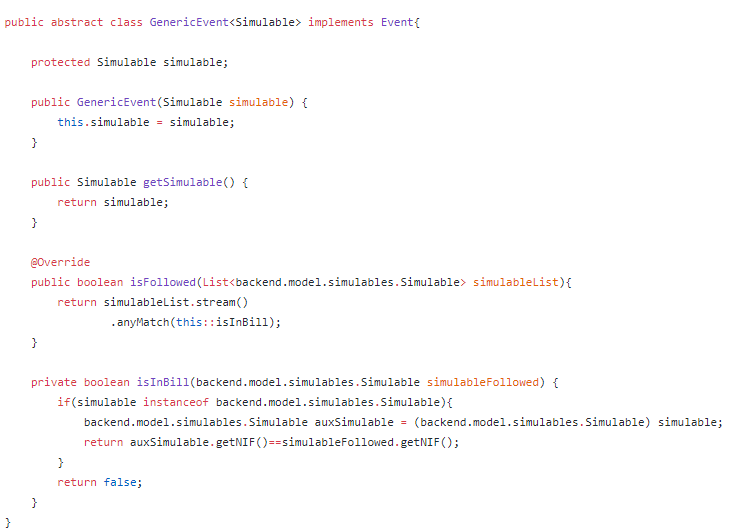


Interfaz Evento (Event)

Los eventos son de tres tipos:

* Eventos de Día: este evento se produce diariamente como su nombre indica. Indica el cambio de día en la simulación. Es el único evento que no involucra ningún simulable. El date de la simulación implementa
* Eventos de Factura o pagos: son eventos que traen consigo facturas de algún tipo o pagos que no generen factura como impuestos de empresas. Las facturas son de facto, eventos. Cada vez que se generan, se añaden también como eventos.
* Eventos de cambios de simulables: son acciones que realizan simulables que provocan algún cambio en las relaciones entre ellos. Por ejemplo, el despido o retiro de algún trabajador, la muerte de un cliente o el cierre de una empresa…. Etc.

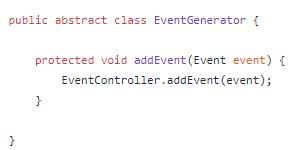
Los eventos de simulables son eventos que no son implementadas por ningún elemento de la simulación, sino que son implementadas en una clase genérica de eventos mas sofisticada, es una clase abstracta que trabaja con un tipo de dato genérico que es siempre un simulable. Cada implementación de esta funciona de forma genérica, es decir, todos los eventos que extienden del **Evento Genérico** tienen un simulable genérico y implementa el método isFollowed (apartado 7.3). A continuación, podemos ver la clase GenericEvent (Evento Genérico) para más detalles.



Clase GenericEvent (Evento Genérico)

Estos eventos solo pueden generados por clases que hereden de la clase EventGenerator (Generador de Eventos). Las clases que pueden generar eventos son las siguientes:

* TimeLine (Línea de Tiempo): para generar los eventos de Día.
* Banco: para generar eventos de pagos de impuestos (Eventos de pagos).
* SimulationIOController (Controlador de Entrada/Salida): para generar eventos de entrada de simulables.
* SimulationCommitter (Committer de la Simulación): para poder generar eventos de cambios de simulables y solicitudes de salida.
* CFDIBillGenerator (Generador de Facturas CFDI): cada vez que se genere una factura, añadirá como evento la propia factura.



Clase EventGenerator (Generador de Eventos)

Los eventos que se generan son administrados por el **Controlador de Eventos**. Los generadores de eventos al añadir un evento añaden directamente el evento al controlador. Este contiene una lista de eventos que se han producido. Cuando se añade, también es mostrado en la consola para poder ver los resultados cuando se ejecuta localmente con el **Interruptor del Simulador** (apartado 7.2.4.1). Además, esta lista es solicitada por el websocket de eventos (puerto de eventos) para ser mostrados en el entorno, esto como es habitual, se explicará con detalle durante el apartado del Entorno de Usuario (7.3). El controlador de eventos solo enviará al puerto los eventos que seguían relevantes para el usuario (7.3). Tras ser proporcionados al websocket, este vacía la lista, para así tener solo los eventos pendientes por mostrar.



Clase EventController (Controlador de Eventos)

### **7.2.7 Capa de Datos**

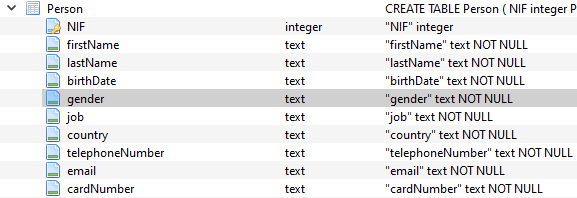
Durante esta sección, se explicará la organización de la base de datos el acceso a esta. El sistema de base de datos que uso en este proyecto es SQLITE Browser (10). La API de acceso a la base de datos es JDBC que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java, independientemente del sistema operativo donde se ejecute o de la base de datos a la cual se accede, utilizando el dialecto SQL del modelo de base de datos que se utilice (12).

A partir de esta API, he desarrollado una implementación propia que envuelve el uso de esta API mediante un conjunto de clases para leer, crear, actualizar y eliminar tablas y filas. El motivo es la disminución de dependencias externas y la motivación de desarrollar un sistema completamente desde cero solo por mí.

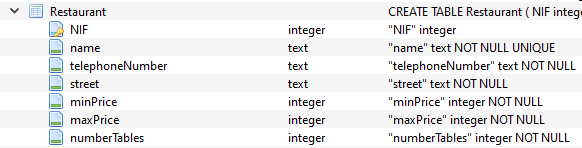
#### **7.2.7.1 Base de datos de la Simulación**

La base de datos está compuesta por 10 tablas, cada uno son accedidas para obtener datos importantes para el desarrollo de la simulación. Las tablas son las siguientes:

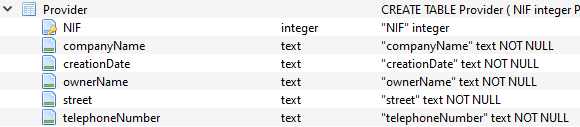
* Person (Persona): Es la tabla de la se extraen los clientes y trabajadores. Tiene 100.000 filas. A continuación, se pueden visualizar los campos.



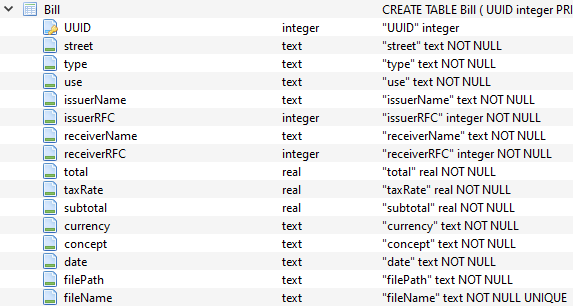
* Restaurant (Restaurante): Tabla con los datos de todos los restaurantes de Gran Canaria extraidos de Tripadvisor.



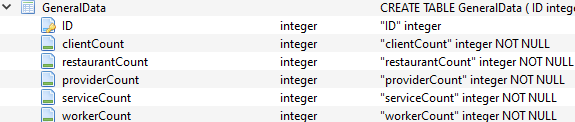
* Provider (Proveedor): De esta tabla se extraen los datos, tanto de los proveedores de products, como los proveedores de servicios.



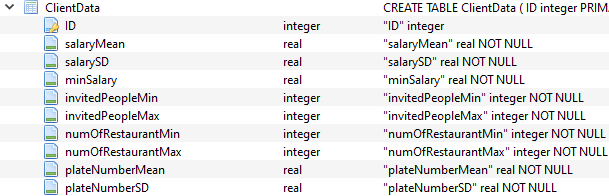
* Bill (Factura): En esta tabla se pueden guardar las facturas generadas si es necesario, ya que ya están generadas en archivos XML en el ordenador, y guardar miles de facturas por segundo en la base de datos puede provocar un cuello de botella a la hora de leer datos para añadir un nuevo simulable, ya que la conexión es única.



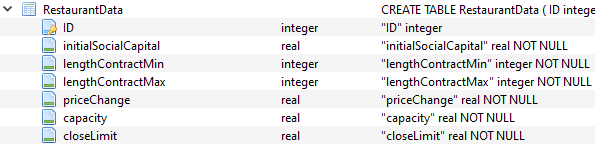
* GeneralData (Datos Generales): Datos por defecto los ajustes generales (7.2.4.5). Tiene una sola fila con los datos por defecto.



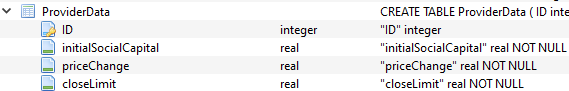
* ClientData (Datos de Cliente): Datos predeterminados para los ajustes de clientes (7.2.4.5). Tiene una sola fila con los datos por defecto.



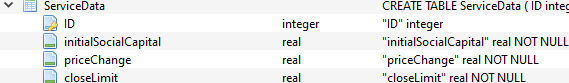
* RestaurantData (Datos de Restaurante): Datos por defecto de los ajustes de restaurantes (7.2.4.5). Tiene una sola fila con los datos por defecto.



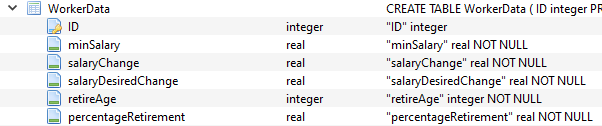
* ProviderData (Datos de Proveedor): En esta tabla se guardan los datos por defecto de los proveedores (7.2.4.5). Tiene una sola fila con los datos por defecto.



* ServiceData (Datos de Servicio): Datos por defecto para los servicios y sus empresas que lo ofrecen (7.2.4.5). Tiene una sola fila con los datos por defecto.



* WorkerData (Datos de Trabajador): Datos predeterminados de los trabajadores (7.2.4.5). Tiene una sola fila con los datos por defecto.



#### **7.2.7.2 Paquete de acceso de datos**

Esta implementación propia me permita acceder a cualquier tabla de la base de datos con una misma implementación. El funcionamiento de este paquete radica en ocultar la implementación de la API y con ello, la necesidad de mandar instrucciones SQL directamente, gracias a una interfaz genérica.

##### **Elementos Básicos de Bases de Datos**

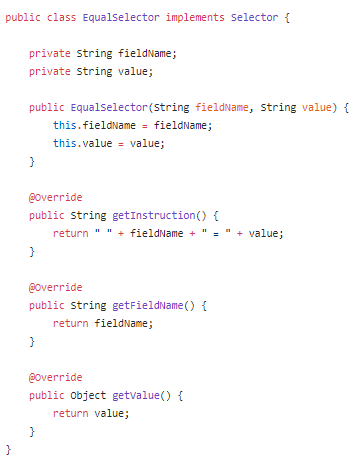
Para poder llevar esta implementación a cabo, y que la interfaz sea genérica y oculte los detalles de la API, es necesario crear ciertos elementos propios de las bases de datos que sepa administrar el paquete de la implementación. Los elementos son los siguientes:

* DataType o tipo de Dato: es una clase con los tipos de datos que se usan en esta simulación de SQLITE, estos son integer (entero), real y text (texto).
* Restricción: esta clase especifica el tipo de restricción que tendrá el dato que referencie. Los usados en esta simulación son Primary Key (Clave Primaria), not null (no nulo) y not null unique (no nulo único).
* Field o campo: es un campo de la tabla. Está compuesto por un tipo de dato y una restricción.
* Header o cabecera: es la cabecera de la tabla, con los todos los datos importantes de la tabla. Contiene el nombre de la tabla, un mapa con clave el nombre del campo y valor el propio campo y el valor inicial de la clave primaria.



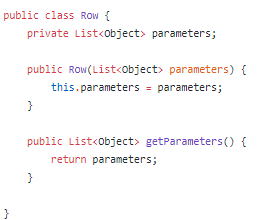
Clase Header (Cabecera)

* Selector: los selectores sirven para buscar datos en la base de datos, en este paquete se proporcionan 3 tipos, de igualdad, “like” y order by. El primer tipo sirve para buscar filas que tengan el campo proporcionado con valor igual al valor proporcionado (edad = 25), el segundo es similar, pero sirve para buscar filas que contengan ese valor (nombre like “Al”), y el último sirve para ordenar las filas finales, ascendente o descendente. Se puede usar tantos como se quieran.



Clase equalSelector (Selector de igualdad)

* Fila: es una fila de una tabla, una lista con los datos pertenecientes a una fila de una tabla.

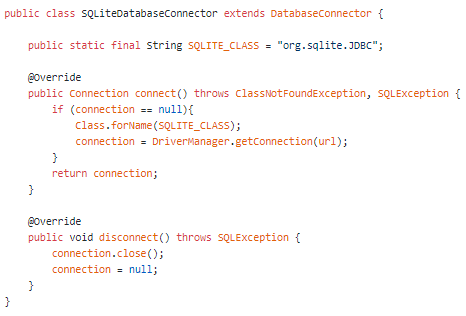


Clase Row (Fila)

##### **Controladores**

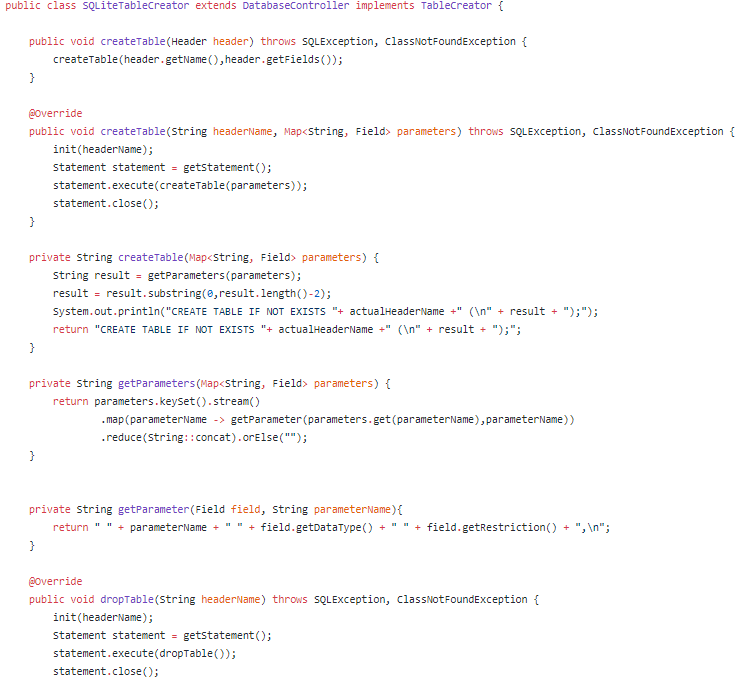
Son los intermediarios directos entre la base de datos y la simulación. Permiten la lectura, creación, eliminación, actualización e inserción en tablas. Los controladores son los siguientes:

* Conector: conector de la base de datos para poder interactuar con este, este conector es único y no se pueden crear varios. Todos los demás controladores los usan.



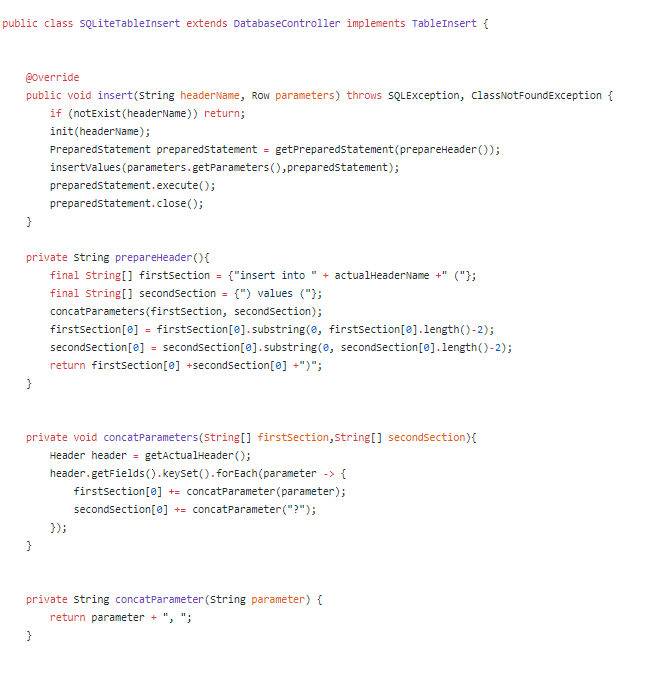
Clase SQliteDatabaseConnector (Conector de la base de datos)

* Creador de tablas: permite crear y destruir tablas en la base de datos de la simulación. Se proporciona una header o cabecera para poder crearla o destruirla.



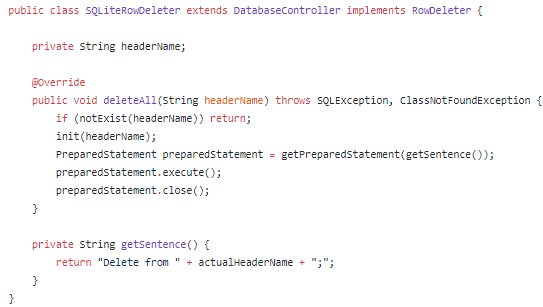
Clase SQLiteTableCreator (Creador de tablas)

* Insertor de Filas: permite insertar filas en las tablas. Se proporciona en nombre de la tabla y la fila usando el elemento explicado.



Clase SQLiteTableInsert (Insertor de filas)

* Eliminador de Filas: permite eliminar filas de una tabla. En este caso, al solo necesitarse el delete all (eliminar todas), solo fue implementado este método, pero la estructura permite añadir nuevas funcionalidades al paquete fácilmente. Se proporciona el nombre de la tabla.



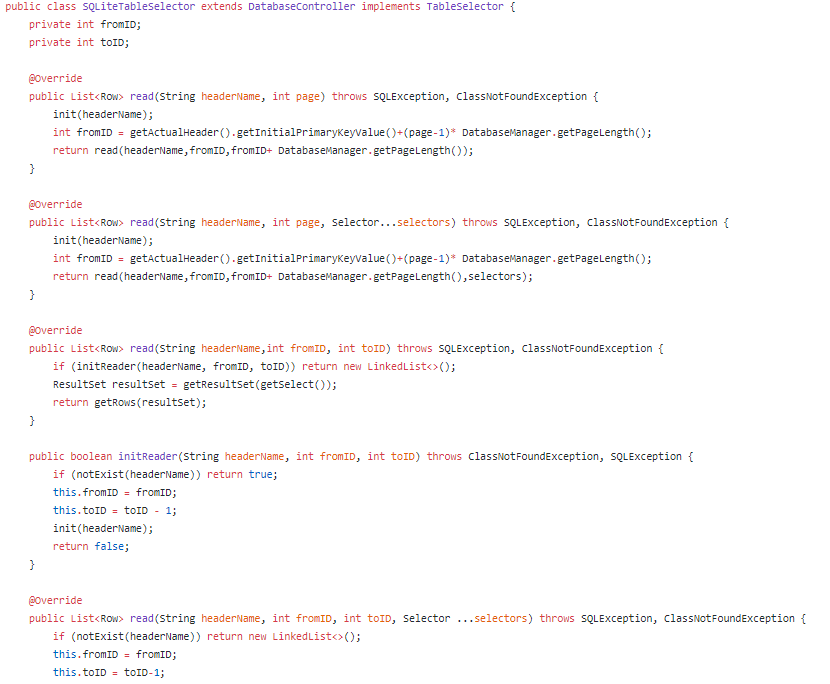
Clase SQLiteRowDeleter (Eliminador de Filas)

* Actualizador de Filas: permite actualizar campos de filas de una de las tablas. Se proporciona el nombre de la tabla, unos selectores de igualdad para saber los campos que hay que actualizar y una lista de selector para proporcionar las condiciones de actualización.



Clase SQLiteRowUpdater (Actualizador de Filas)

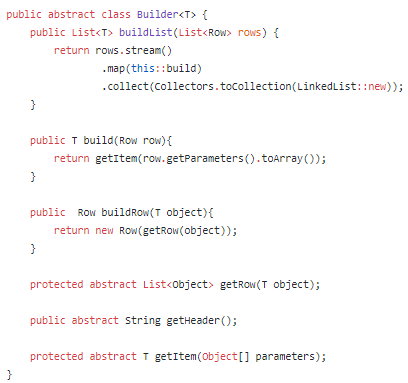
* Selector de filas: permite seleccionar filas y obtener los datos en la simulación para su posterior uso. Se proporciona el nombre de la tabla y un rango de valores (inicio y final) con respecto a los valores de la clave primaria.



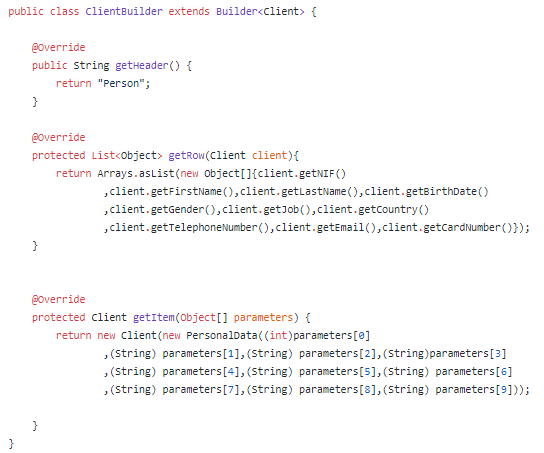
Clase SQLiteTableSelector (Selector de filas)

##### **Builders**

Para la selección e inserciones de filas (Entrada/Salida) se utiliza un traductor que traduce las Filas a objetos de la simulación y viceversa. Hay un traductor o constructor por cada objeto que se quiere guardar en la simulación, es decir, uno para clientes, restaurantes, proveedores… etc. Estos constructores se llaman builders. A continuación, tenemos la clase genérica de un builder y un ejemplo de su implementación.

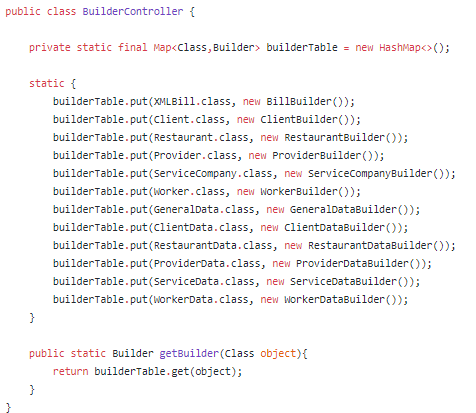


Clase Builder genérica



Clase Builder de cliente

Además, hay un controlador de builders que tiene un mapa de builder para que cuando se quiera obtener o insertar datos de alguna entidad de la simulación (cliente, restaurante…), el controlador te proporcione el builder adecuado sin conocer que implementación es, utilizando únicamente los métodos genéricos para traducción de fila a objeto y viceversa. A partir de la Class (clase) de la entidad (Client, Provider, Restaurant…), se obtiene el builder.



Clase BuilderController (Controlador de Builders)

##### **Manager de la Base de Datos**

Guarda las cabeceras de todas las tablas que hay en la simulación para mejorar el rendimiento y la facilidad de uso del paquete. Además, proporciona métodos de paginación para el acceso a los datos.



Clase DatabaseManager (Manager de la base de datos)

##### **Administrador de Tablas**

Es una clase que engloba todos los controladores en una clase que proporciona todos los servicios de todos los controladores desde un mismo punto. Además, mediante el uso del controlador de builders permite acceder a la base de datos sin conocer ningún elemento o implementación del paquete. Cada elemento de la simulación que requiere de acceder a la capa de datos tiene una instancia de esta clase, como el **Inicializador de Simulables** (Apartado 7.2.4.3).



Clase SQLiteTableAdministrator (Administrador de Tablas)

##### **Resultado del Paquete Proporcionado**

Como resultado, se obtiene una interfaz propia, sencilla y genérica de acceso de datos que esconde y separa completamente la implementación del resto de la simulación. Los elementos de la simulación que quieran acceder a los datos solo deben tener una instancia del **Administrador de Tablas** y a la hora de obtener un simulable solo tienen que proporcionar la clase de este simulable, el valor inicial (que procede del NIF del simulable como vimos anteriormente o es solo una fila en el caso de los ajustes) y la cantidad de simulables que se quiere de ese tipo.

#### **7.2.7.2 Otras Opciones de Acceso a Datos**

Todas las formas de acceso a datos,incluido en la base de datos, contienen una interfaz genérica de acceso pero con implementaciones distintas. Además de la lectura de la base de datos existen dos formas de acceso adicionales que se han usado durante el desarrollo de este proyecto en versiones anteriores, lecturas de archivos CSV, y lectura directa de internet (Web Scrapping).

* CSV: los datos de clientes-trabajadores y proveedores-servicios inicialmente eran leidos de archivos CSV (Comma-Separated Values) en el que cada fila es una instancia del simulable. Este archivo provino del generador de datos aleatorios usado (6).
* Jsoup: mediante la librería de Jsoup(5) se obtenía los datos al principio de los restaurantes. Esta clase leía los datos de restaurantes de la cantidad solicitada a través de la lectura directa de la página web de Tripadvisor, concretamente los restaurantes de Gran Canaria.

### **7.2.8 Preparación de Simulables**

Tras la lectura y creación de los simulables, estos no están todavía preparados para entrar en la simulación. Estos requieren una serie de necesidades adicionales que cubrir. Estas serán proporcionadas por los preparadores de los simulables. Estos preparadores son llamados por el **Inicializador de Simulables**. Los preparadores son clases funcionales, es decir, que siguen la metodología de la programación funcional. Estos mediante la lista de simulables de algún tipo que tienen preparar en algún aspecto, paralelamente cada uno de ellos es preparado individualmente en un hilo aparte. Los inicializadores son los siguientes:

* Inicializador de Rutinas: inicializas las rutinas de todos los clientes que se van a preparar.
* Inicializador de trabajadores: selecciona trabajo a cada uno de los trabajadores.
* Inicializador de personal: selecciona trabajadores para trabajar en el restaurante que se va a inaugurar.
* Inicializador de productos: selecciona el producto que va a vender cada uno de los proveedores a preparar.
* Inicializador de provisión de productos: añade los proveedores necesarios a las empresas que quieren entrar en la simulación.
* Inicializador de servicios: selecciona el servicio que se ofrecerá por esta compañía.
* Inicializador de provisión de servicios: añade los servicios necesarios a las empresas que quieren entrar en la simulación.

### **7.2.9 Resumen**

Este módulo es el núcleo del proyecto, el punto central en el que se generan todos los procesos principales en un entorno financiero. Este núcleo es independiente y sin ninguna interfaz de usuario, ya que es el objetivo del otro módulo. El núcleo funciona de forma autónoma, ya que solo necesita ser inicializado, y este generará los agentes correspondientes para arrancar la simulación y que cada uno de los participantes participe en este entorno con el rol o papel dado.

En la siguiente sección se explicará el siguiente módulo encargado de mostrar al usuario los eventos o acciones que se están produciendo internamente en la simulación, además de poder interactuar con la simulación y cambiar los ajustes principales de esta para tener unos resultados u otros.

## **7.3 Entorno de usuario de la Simulación**

### **7.3.1 Introducción al entorno**

La simulación hasta ahora no ha mostrado ninguna interfaz para mostrar la simulación y los eventos que se producen al usuario. Este es el objetivo de esta sección, mostrar de cierto modo la simulación al usuario.

El entorno se ha desarrollado con en web con una arquitectura cliente-servidor, en el que el servidor usa directamente la simulación, ejecutándole e interactuando acorde a las instrucciones solicitadas desde el cliente.

En cuanto al servidor, se ha usado el Tomcat EE Server con el lenguaje en Java (13). Este servidor funciona de forma autónoma a partir de un Servlet o archivo JSP, como archivo inicial que ejecutar. En este caso, ejecuta un Servlet que se explicará más adelante. El servidor está compuesto por dos Servlet y una serie de comandos que controlan la simulación a partir de las instrucciones mandadas desde el cliente.

En cuanto al cliente, se ha desarrollado usando archivos JSP, JavaScript y CSS. Los archivos JSP se usan para mostrar el contenido al alcance del usuario, los archivos JavaScript para controlar los procesos que se producen en el cliente y los archivos CSS para generar una vista atractiva al usuario final.

### **7.3.2 Servidor**

#### **7.3.2.1 Servlets**

##### Inicializador de la Simulación

Inicializa las URIs de los distintos archivos como la base de datos, los archivos CSV o la carpeta donde están las facturas. Además, vacía las facturas de la base de datos si hay, instancia los **Datos de la Simulación** y guarda los datos de los ajustes en la sesión de la web. Por último, carga la vista principal del cliente, que se explicará en la sección del cliente (7.3.3).



Clase InitSimulationServlet (Inicializador de la Simulación)

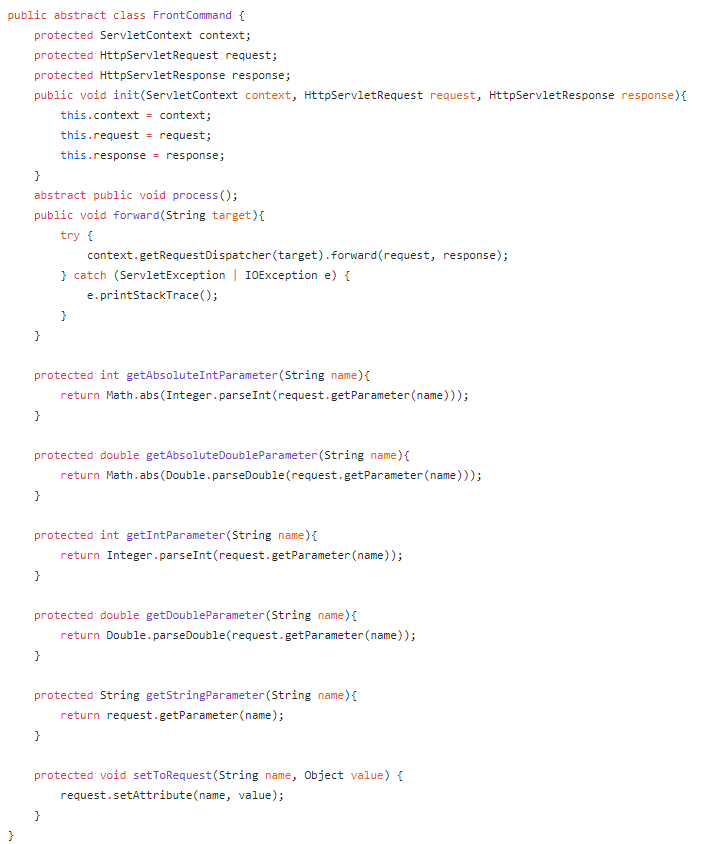
##### FrontController

Corresponde a un patrón de diseño conocido, que consites en un usar un Servlet como el receptor centrar de peticiones HTTP en el que se me mande el comando a solicitar a través de la request. A partir de ahí, todas las acciones del cliente hacen una petición al mismo Servlet con el comando a realizar y esta llama al comando directamente. Por ello a continuación, se explicarán todos los comandos que controla el FrontController.



Clase FrontControllerServlet (FrontController)

El FrontController usa un tipo de comando llamado FrontCommand, que es una clase abstracta de la que todos los comandos que se usarán heredan de esta clase implementando el método process(), que como su nombre indica, procesa el comando elegido.

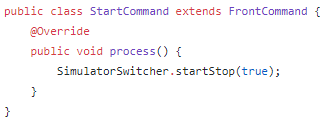


Clase Abstracta FrontCommand

#### **7.3.2.1 Comandos**

Los tipos de comandos que se pueden encontrar en este proyecto son los siguientes:

* UnknownCommand: este comando se ejecuta cuando no se ha encontrado ningun comando con el nombre enviado en la petición. Se muestra una vista indicando el error producido.
* StartCommand/RestartCommand: empieza, termina o reinicia la simulación.
* Comando de paginación: comando abstracto que extiende del Frontcommand proporcionando una interfaz a implementar para paginar una lista de datos que se mostraran posteriormente en una tabla. Se utiliza para mostrar en tablas paginadas los distintos simulables, por lo que cada uno de ellos tiene una implementación de este comando para mostrarse en el cliente con su vista adecuada.
* SearchCommand: permite buscar simulables, mostrándolos directamente en una tabla como resultado de la petición.
* FollowCommand/UnfollowCommand: te permite seguir o dejar de seguir simulables. Esto se explicará durante la explicación de las funcionalidades en la sección del cliente (7.3.3).
* DownloadCommand: te permite descargar una factura seleccionada. Se proporcionada el UUID de la factura (ID) para luego proporcionar al cliente el archivo XML de la factura.
* Comando de Ajustes: son una larga lista de comandos para controlar los ajustes, ya sea mediante la actualización de los ajustes, el guardado de estos, la cancelación de cambios, poner los ajustes por defecto… etc.



Ejemplo de comando, StartCommand

### **7.3.3 Cliente**

# **8. Conclusiones**

## **8.1 Resultado Obtenido**

### **8.1.1 Del Proyecto**

El resultado que se ha obtenido es un proyecto compuesto por dos módulos: el núcleo de procesos, y la interfaz que muestra al usuario los procesos internos. El núcleo cumple por si solo y de forma independiente con los objetivos del proyecto: generar facturas realistas a través de un entorno financiero realista. El objetivo del segundo módulo es mostrar los eventos producidos internamente en el núcleo sin interferir directamente en la simulación ni en su rendimiento, además de proporcionar una interfaz de interacción con la simulación directa.

El resultado obtenido del primer módulo corresponde con los objetivos iniciales proporcionando adicionalmente, una arquitectura adecuada y correctamente desarrollada para ampliar fácilmente el proyecto manteniendo la arquitectura actual, y añadiendo, sin demasiadas dificultades, nuevos elementos a la simulación.

En el caso del segundo módulo, cumple también con los objetivos, ya que proporciona una interfaz sencilla y clara para mostrar los eventos que se producen en la simulación, además de proporcionar una gran variedad de ajustes e interacciones para poder adecuar las necesidades a las situaciones requeridas o para adaptarse al estado actual del mundo como, por ejemplo, reduciendo el porcentaje de ocupación de los restaurantes debido a una pandemia.

### **8.1.2 A Nivel Personal**

A nivel personal, este proyecto ha supuesto un gran avance a nivel organizativo, ya que me he tenido que organizar para hacer un proyecto a largo plazo, el mas largo hasta ahora. Este proyecto empezó en octubre y terminó en julio, 9 meses de trabajo constante para desarrollar este proyecto, lo que me ha llevado a tener que organizarme adecuadamente para cumplir con los objetivos de las distintas iteraciones de forma completamente autónoma.

Además, en este proyecto he sido mi propio jefe, con el apoyo y el consejo de mi tutor, pero, a fin de cuentas, un proyecto desarrollado de forma independiente y tomando yo la mayoría de las decisiones necesarias para continuar con el proyecto y para solucionar los diversos problemas que surgían.

Ha sido un arduo trabajo que ha llevado muchas horas pero que ha tenido sus frutos donde he podido aprender a ser mas independiente, mas organizativo y mas profesional.

### **8.1.3 A Nivel Laboral**

A nivel laboral también ha supuesto un enorme avance, tanto en el uso de nuevas tecnologías, como en el desarrollo de una arquitectura adecuada para la escalabilidad de la simulación.

## **8.2 Aportaciones**

Este proyecto aporta un generador sintético de datos financieros que puede ser usado para el desarrollo de otros proyectos financieros de Big Data, como el desarrollado en este proyecto conjunto. Además, permite simular distintos puntos de vistas o situaciones en las que se puede encontrar el entorno financiero, adaptando los ajustes a las nuevas necesidades, con lo que se puede obtener resultados de simulaciones reales que tengan en cuentas estas necesidades. Como resultado, se pueden hacer estudios razonables de la situación que podría encontrarse la economía en un futuro con esas características o necesidades, causadas por alguna pandemia o problema a nivel mundial como el Coronavirus actual.

Además, se aporta una arquitectura bien desarrollada para poder ser adaptada a nuevos elementos y eventos que afecten a la simulación, con lo que poder incrementar el realismo de este indefinidamente. Es un proyecto escalable indefinidamente, ya que el entorno empresarial y la economía es un sector inmenso y complicado que se puede reflejar adecuadamente a través de iteraciones que incrementen gradualmente el realismo con la economía de la realidad.

## **8.3 Trabajos Futuros**

Unas de las ventajas que tiene este proyecto, es su infinita escalabilidad, debido al tamaño y a la complejidad del sector. Por ello, se pueden extender el proyecto en una gran variedad de direcciones. Podemos destacar las siguientes mejoras:

* **Inversiones**: Un aspecto importante en la economía que carece actualmente el proyecto y sería interesantes añadir son las inversiones. Estas inversiones se producen para mejorar la calidad del producto o servicio ofrecido, como la compra de nuevo locales, mejoras de la infraestructura a usar… etc.
* **Extensión a nuevos sectores**: este proyecto se ha centrado en el sector de la restauración debido a la complejidad del sector de la economía y negocios. Un gran avance sería extender este proyecto a sectores relacionados a la restauración, como el sector turístico o el de la construcción.
* **Añadir más interacciones y factores**: este proyecto se ha centrado en este punto añadiendo el máximo de realismo a esta simulación. Aun así, se puede seguir numerosas mejoras como la adición de factores como los días festivos o las vacaciones, así como añadir un gobierno y una hacienda que controle la economía como un agente más.
* **Mejorar la comunicación Simulación-Usuario**: actualmente en el proyecto, la simulación se comunica con el usuario a través del entorno de usuario, usando un panel que muestra en texto todos los eventos que se han producido relacionados con los agentes que se siguen. Además, se muestran otros datos importantes como el número de agentes de cada tipo. Sin embargo, se pueden mejorar la simulación añadiendo mas interacciones directas con simulables concretos que se seleccionen o añadir nuevos datos genéricos que se muestren al usuario, como el porcentaje de paro de los trabajadores o el precio medio de un producto.

# **9. Bibliografía**

1. La facturación

<https://www.mheducation.es/bcv/guide/capitulo/8448614194.pdf>

2. CFDI

<https://es.wikipedia.org/wiki/CFDI>

3. Ejemplos CFDI

<https://facturador.zendesk.com/hc/es/articles/115012733308-Ejemplo-de-CFDI-Emisi%C3%B3n-3-3>

4. Tutorial Básico Ajax con JSP

<https://jarroba.com/ajax-con-jsp-y-servelts/>

5. Librería Jsoup

<https://jsoup.org/>

6. Generador de datos Aleatorios

<https://www.onlinedatagenerator.com/>

7. Fintech

<https://es.wikipedia.org/wiki/Tecnolog%C3%ADa_financiera>

<https://www.condusef.gob.mx/Revista/index.php/usuario-inteligente/educacion-financiera/763-que-son-las-fintech>

8. Jquery

<https://jquery.com/>

9. Tripadvisor

<https://www.tripadvisor.es/Restaurants-g187471-Gran_Canaria_Canary_Islands.html>

10. DB Browser for SQLite

<https://sqlitebrowser.org/>

11. ThreadPoolExecutor

<https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ThreadPoolExecutor.html>

12. JDBC

<https://www.ibm.com/support/knowledgecenter/es/SSGU8G_11.70.0/com.ibm.jdbc_pg.doc/ids_jdbc_011.htm>

13. Tomcat

<https://tomee.apache.org/>

14. Arquitectura Publisher/Subscriber

<http://arquitecturasomos4.blogspot.com/2014/12/publish-subscriber.html>

15. Sistema Multi-Agente

<https://es.wikipedia.org/wiki/Sistema_multiagente>

16. Bytecode Java

<https://es.wikipedia.org/wiki/M%C3%A1quina_virtual_Java>