Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Computación

Aplicación web para la gestión de carteras de inversión usando técnicas de Inteligencia Artificial

Autor: Víctor Arranz Barcenilla

Tutor: Valentín Cardeñoso Payo

«Una mente necesita de libros igual que una espada de una piedra de amolar.»

Tyrion Lannister

Agradecimientos

Han sido muchas personas las que han ayudado a que este proyecto haya salido adelante .... En primer lugar agradecérselo a mis tutores .... ya que con sus constantes revisiones y correcciones tanto la aplicación como la memoria ha ido por el camino adecuado.

A este grupo, a aquel otro ...

Resumen

Aquí deberá ir el resumen del trabajo, cuyo esquema (una frase para cada parte) podría ser: MOTIVACION: OBJETIVO: TAREAS: RESULTADOS: CONCLUSION:

Abstract

The same 'Resumen' but in English, please ...

Índice general

Índice de cuadros

Índice de figuras

Todo list

Capítulo 1

Introducción

Al principio, está todo por hacer.

Sobre cómo realizar un TFG, la memoria y todas esas cosas, se puede consultar el libro de García[1]. Suerte!

Por otra parte, las órdenes Linux aparecen siempre así: `$ ls -al`

1.1 Introducción

Nuestra visión del problema, la necesidad, el por qué es interesante hacer esto ...

1.2 Motivación

Qué nos lleva a plantearnos hacer este proyecto ...

1.3 Objetivos

Los objetivos son logros, no lo olvides.
Nuestro objetivo principal .....
Si hubiese, los secundarios ....

Capítulo 2

Planificación

2.1 Metodología

Todo trabajo requiere un método bien especificado y establecido. A lo largo del tiempo se han ido desarrollando diferentes metodologías de desarrollo con el objetivo de organizar el trabajo en equipo de un grupo de personas y de llevar a cabo las tareas requeridas de un proyecto de una manera productiva y eficaz [referencia].

Pese a que a lo largo de la carrera siempre se han utilizado metodologías tradicionales en los procesos de desarrollo de software, como son el desarrollo iterativo, en cascada o incremental, en el presente trabajo se ha optado por dar un enfoque más ágil. Esto suele ser más habitual en los equipos de trabajo de la industria, debido a la alta flexibilidad y agilidad que permiten este tipo de metodologías. Este enfoque tiene a su vez la ventaja de permitir la adaptación del producto y de los subproductos que se fabrican a las necesidades que van surgiendo durante el proceso de desarrollo, así como la construcción de equipos de trabajo autosuficientes e independientes, que se coordinan mediante reuniones periódicas. Los métodos ágiles tienen como base el desarrollo incremental de las metodologías tradicionales, buscando agregar unas pocas nuevas funcionalidades al producto en cada ciclo de desarrollo, siendo estos de duración breve (como mucho de ocho semanas de duración, siendo dos o cuatro semanas la duración habitual). Las principales metodologías ágiles son las siguientes:

- **Kenban:** consiste en dividir las tareas en tres bloques: tareas finalizadas, tareas en curso y tareas pendientes, creando un flujo de trabajo muy claroq ue permite incrementar el valor del producto.
- **Scrum:** similar al anterior, los ciclos de iteración son cortos y están fijados antes de comenzar el proyecto. Se introduce el concepto de sprints[referencia], que es la forma de denominar en esta metodología a cada iteración o ciclo de trabajo. En cada sprint se ha de generar lo que se denomina un entregable, es decir, un incremento que aporte valor al cliente. Es importante recalcar que en cada sprint, el producto ha de ser funcional para el cliente. Cada sprint de Scrum está formado a su vez por varias fases:
  - **Planificación:** qué se va a hacer en el sprint y cómo se pretende eso llevar a cabo.
  - **Scrum diario:** reuniones diarias de duración corta donde los miembros del equipo exponen sus progresos y dificultades.
  - **Revisión:** se acepta o no el sprint realizado.
  - **Retrospectiva:** se analiza cómo ha ido el sprint, qué problemas ha habido y cómo mejorarlos.

- **Lean:** busca que pequeños equipos de trabajo con alta capacitación desarrollen cualquier tipo de tarea en poco tiempo. Se centra en las personas, dejando en segundo plano el tiempo y los costes.
- **Programación Extrema (XP):** centrado en las relaciones interperpersonales donde habitualmente se realiza programación por parejas. Se basa en principios como: diseño sencillo, testeo, refactorización, integración continua y entregas semanales entre otros.

En el proyecto llevado a cabo en este trabajo se plantea un enfoque similar a Scrum con sprints de dos semanas pero con matices, ya que al ser un equipo de desarrollo compuesto únicamente por una persona, ciertos elementos de la metodología como las reuniones entre miembros carecen de sentido. A su vez, se incluye en el proyecto cierto carácter de metodología tradicional al llevarse a cabo un desarrollo web con prototipos, lo que no aleja la metodología del enfoque ágil incremental planteado por Scrum. Del mismo modo, al tratarse de un proyecto académico donde parte del trabajo es de investigación y aprendizaje, las primeras semans del mismo no seguirán una metodología ágil como tal al no producirse entregables, puesto que estas primeras semanas se destinarán al despliegue del servidor y al análisis del problema y las prosibles soluciones.

2.2 Fases y costes

Los sprints planteados en este trabajo son quinquenales y se resumen en la siguiente tabla:2.1.

Sprint	**Nombre de actividad**	**Semanas**
Sprint 1	Análisis y aprendizaje de las herramientas a utilizar para el despliegue del servidor web	1 - 2
Sprint 1	Despliegue del servidor web	1 - 2
Sprint 1	Estudio de activos y mercados	1 - 2
Sprint 1	Estudio de modelos de investigación operativa	1 - 2
Sprint 1	Estudio de modelos de inteligencia artificial	1 - 2
Sprint n	Escritura de la memoria del TFG	16 - 17

Cuadro 2.1: Sprints de desarrollo del proyecto previstos.

2.3 Actividades

Una manera habitual de definir las tareas que requiere un sistema software consiste en la creación de un diagrama de descomposición de tareas (WBS). Este enfoque implica identificar las principales tareas requeridas para llevar a cabo la construcción del sistema (tareas de alto nivel) para, posteriormente, descomponer cada tarea en subtareas de nivel más bajo. El modo de proceder consiste en añadir tareas en cada rama"únicamente si están directamente relacionadas en la consecución de la tarea "padre". Cada rama debe descomponerse, al menos, hasta un punto en el cual la "hoja"pueda asignarse a un único individuo o sección de una organización. En el presente trabajo, todo será llevado a cabo por un mismo individuo pero se muestra la descomposición que habría que hacer en el ámbito profesional de una organización.

Un aspecto importante a considerar en este tipo de diagramas es el nivel de dtealle que se quiere mostrar, ya que demasiada profundidad puede dar lugar a un número de tareas que sea complicado de gestionar, mientras que un diagrama demasiado superficial proporciona un nivel de detalle demasiaod escaso para el necesario control de proyecto.

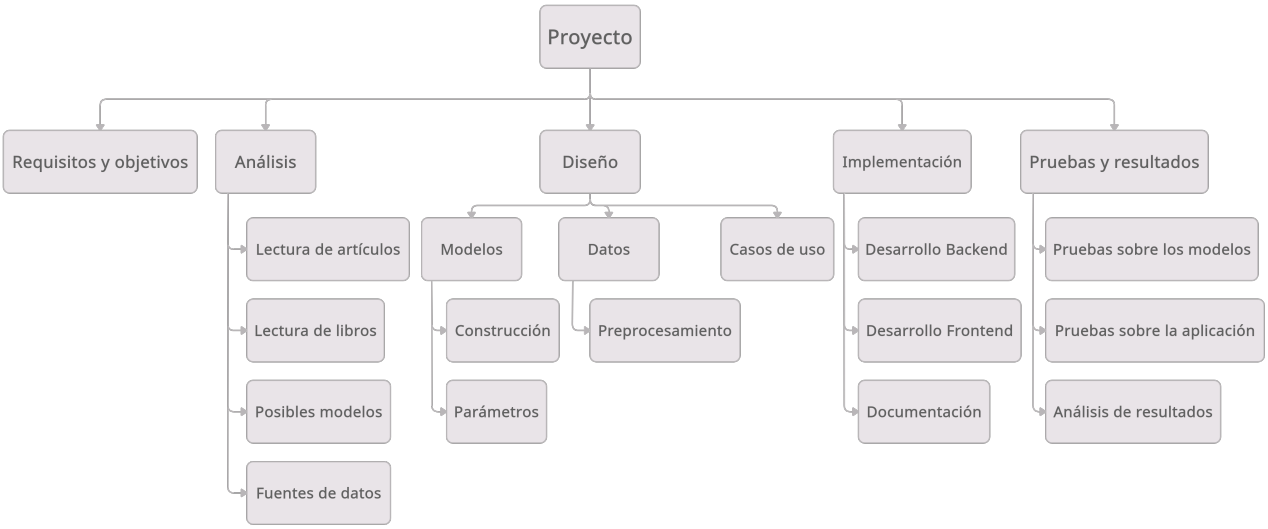


Figura 2.1: Diagrama de tareas.

2.4 Recursos

Los recursos requeridos en este proyecto software son únicamente de dos tipos:

- **Humanos:** el alumno encargado de desarrollar el proyecto en su totalidad.
- **Tecnológicos:** el equipo de trabajo que utilizará el desarrollador, la máquina virtual que alojará la aplicación (proporcionada por la Escuela de Ingeniería Informática de la Universidad de Valladolid), conexión a internet para porder acceder a dicha máquina y para la comunicación con el tutor del trabajo y luz corriente para poder utilizar los equipos informáticos.

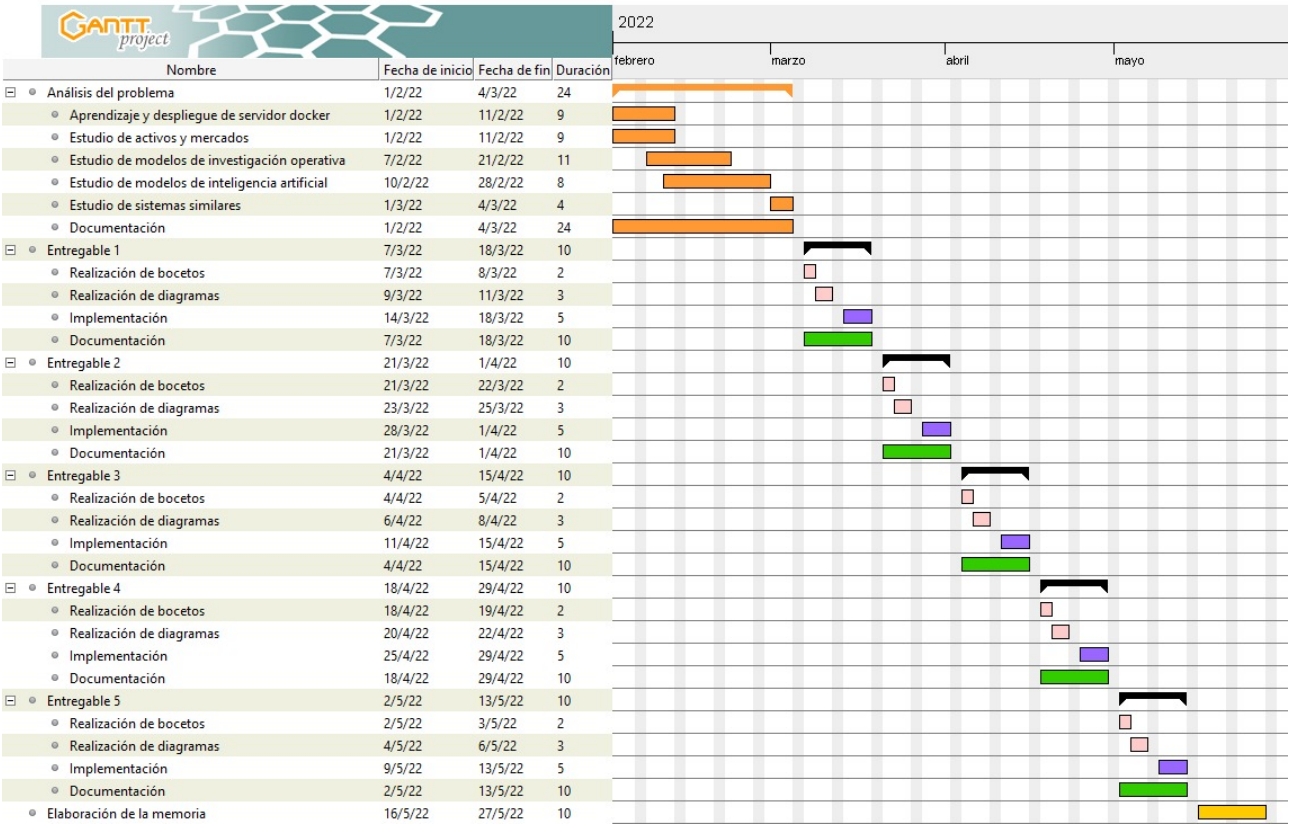
2.5 Planificación inicial

De cara a la realización de este proyecto y conforme a la metogología anteriormente explicada, podemos agrupar las tareas que conformarán el trabajo en cuatro grandes bloques:

- **Análisis del problema:** en esta fase se estudiará el problema a resolver en este trabajo, tanto desde el punto de vista conceptual como desde el punto de vista práctico. Se llevarán a cabo tareas de aprendizaje, despliegue del servidor que alojará la aplicación web, lectura de diversos artículos, papers y libros que permitirán trazar la hoja de ruta sobre los modelos a implementar en las fases posteriores. También se analizarán las tecnologías a utilizar en el proyecto y el modo de utilizarlas para la implementación del sistema.
- **Diseño del sistema:** se llevarán a cabo tareas de ingeniería de software que tratarán de definir y especificar claramente la estructura y funcionamiento de la aplicación web. Para ello se planteará un diseño con mockups y los diagramas necesarios para explicar todo lo necesario del sistema.
- **Implementación del sistema:** programación de los diferentes módulos que componen la aplicación web diseñada en la fase anterior.
- **Elaboración de la memoria:** escritura final de la memoria, revisando todo lo redactado anteriormente en las tareas de documentación.

Para adaptar estas tareas a la metodología explicada en el punto 2.1 se ha de tener en cuenta que la primera fase de desarrollo de este trabajo englobará toda la parte de análisis, que no se incluirá como tal en el marco ágil elegido para el desarrollo del trabajo. El resto del trabajo seguirá el enfoque comentado, teniendo en cuenta que las tareas de diseño e implementacion se realizarán en cada sprint produciendo entregables de manera incremental y que la tarea de escritura de la memoria constituirá en sí el último sprint donde el entregable producido es la memoria final del proyecto.

El siguiente diagrama de Gantt muestra el diagrama de tareas del proyecto, teniendo en cuenta que el color naranja identifica a la fase inicial de análisis, el color rosa identifica las tareas de diseño, el color morado las tareas de implementación, el verde la documentación que se va realizandon en paralelo al resto de actividades de cada sprint y, finalmente, el color amarillo indica la escritura final de la memoria.



Nombre	Fecha de inicio	Fecha de fin	Duración
Análisis del problema	1/2/22	4/3/22	24
Aprendizaje y despliegue de servidor docker	1/2/22	11/2/22	9
Estudio de activos y mercados	1/2/22	11/2/22	9
Estudio de modelos de investigación operativa	7/2/22	21/2/22	11
Estudio de modelos de inteligencia artificial	10/2/22	28/2/22	8
Estudio de sistemas similares	1/3/22	4/3/22	4
Documentación	1/2/22	4/3/22	24
Entregable 1	7/3/22	18/3/22	10
Realización de bocetos	7/3/22	8/3/22	2
Realización de diagramas	9/3/22	11/3/22	3
Implementación	14/3/22	18/3/22	5
Documentación	7/3/22	18/3/22	10
Entregable 2	21/3/22	1/4/22	10
Realización de bocetos	21/3/22	22/3/22	2
Realización de diagramas	23/3/22	25/3/22	3
Implementación	28/3/22	1/4/22	5
Documentación	21/3/22	1/4/22	10
Entregable 3	4/4/22	15/4/22	10
Realización de bocetos	4/4/22	5/4/22	2
Realización de diagramas	6/4/22	8/4/22	3
Implementación	11/4/22	15/4/22	5
Documentación	4/4/22	15/4/22	10
Entregable 4	18/4/22	29/4/22	10
Realización de bocetos	18/4/22	19/4/22	2
Realización de diagramas	20/4/22	22/4/22	3
Implementación	25/4/22	29/4/22	5
Documentación	18/4/22	29/4/22	10
Entregable 5	2/5/22	13/5/22	10
Realización de bocetos	2/5/22	3/5/22	2
Realización de diagramas	4/5/22	6/5/22	3
Implementación	9/5/22	13/5/22	5
Documentación	2/5/22	13/5/22	10
Elaboración de la memoria	16/5/22	27/5/22	10

Figura 2.2: Diagrama de Gantt de actividades.

2.6 Presupuesto inicial

Teniendo en cuenta los recursos indicados en 2.4 y estimando una duración de unas 300 horas para la realización total del trabajo (12 créditos) repartidas en un total de unos 4 meses, podemos estimar el coste del proyecto. Para ello tenemos que hacer varias consideraciones:

- Estimamos el salario del desarrolador como el salario medio de un ingeniero informático en España que, de acuerdo a [https://www.jobted.es/salario/ingeniero-inform %C3 %A1tico], es de unos 36500€ brutos al año, es decir, unos 18.71€/hora (aunque esto es muy variable en función de la empresa, años de experiencia, etc).

- Estimamos el coste de la luz durante los próximos 4 meses en España utlizando el promedio del coste de la luz a lo largo de 2021, que fue de 111,38 €/MWh de media de acuerdo a [referencia]. Teniendo en cuenta una utilizacion aproximada de 300 horas del equipo de trabajo, que la batería está al 96 %, lo que hace un total de 4542 mAh que, teniendo en cuenta que el votlaje en España es de 230V da un valor de 1044.66 Wh tras hacer la conversión (para cada carga). El total de cargas completas es de unas 100 (300h/3h cada carga), por lo que el total de gasto en luz en MWh se estima en unos 0.104466 WMh.
- Ignoramos el consumo de la máquina virtual al desconocerse este.
- El equipo utilizado para el desarrollo tiene un tiempo de vida menor a un año y es un ordenador MSI GL65 Leopard con un precio actual de unos 1200€ según distintas páginas de venta de productos informáticos.

Esto da como resultado un coste resumido en la siguiente tabla:

Concepto	**Coste**
Horas de trabajo del desarrollador	5613€
Equipamiento	1200€
Desgaste de equipamiento	10€
Luz	1.16€
Total	**6824.16€**

Cuadro 2.2: Costes del proyecto.

2.7 Desviaciones de la planificación inicial

2.8 Coste final

2.9 Análisis de riesgos

En esta sección se tratará de realizar un análisis de los posibles riesgos a los que se encuentra expuesto este proyecto software. Los únicos que se van a considerar son los riesgos de proyecto, ignorándose los posibles riesgos de negocio. Esto se debe a que la aplicación web diseñada en este tfg no se va a crear con la intención de comercializarla.

Para poder realizar un correcto análisis y evaluación de los riesgos de un proyecto, es necesario asignar a cada uno de ellos una probabilidad de ocurrencia y una medida numérica del impacto que tendría el mismo en el caso de materializarse. El enfoque elegido para llevar a cabo esto, es el propuesto por Barry Boehm [referencia], consistente en analizar cualitativamente tanto las probabilidades de ocurrencia como los impactos de los riesgos.

Nivel de probabilidad	**Rango**
Alto	Más de un 50 % de probabilida de ocurrencia
Significativo	30-50 % de probabilidad de ocurrencia
Moderado	10-29 % de probabilidad de ocurrencia
Bajo	Menos de un 10 % de probabilidad de ocurrencia

Cuadro 2.3: Tabla probabilidades Barry Boehm.

Nivel de impacto	Rango
Alto	Más de un 30 sobre el gasto presupuestado
Significativo	20-29 % sobre el gasto presupuestado
Moderado	10-19 % sobre el gasto presupuestado
Bajo	Menos de un 10 % sobre el gasto presupuestado

Cuadro 2.4: Tabla impactos Barry Boehm.

Una vez estudiadas las probabilidades e impactos de los riesgos de un proyecto, conviene definir los planes de actuación frente a los mismos, los cuales se engloban en dos categorías principales:

- **Plan de prevención o protección:** acciones llevadas a cabo con el objetivo de reducir la probabilidad de que un riesgo se manifieste.[referencia]
- **Plan de contingencia:** acciones llevadas a cabo con el objetivo de que los perjuicios causados por la materialización de un riesgo sean lo menos graves posibles. Constituye una guía de actuación a seguir cuando un riesgo se manifiesta y trata de reducir su impacto todo lo posible.

Así pues, los riesgos encontrados en el presente proyecto junto a sus consiguientes planes de prevención y contingencia, quedan resumidos en la siguiente tabla:

Id	Descripción	Probabilidad	Impacto
01	Retraso en la planificación de cualquiera de las tareas.	Significativa	Alto
02	Enfermedad del desarrollador del proyecto	Moderado	Alto
03	Incumplimiento de la planificación debido a un mal planteamiento inicial de la misma	Significativa	Alto
04	Falta de conocimiento de tecnologías	Significativa	Significativo
05	Problemas en la máquina virtual que aloje la aplicación	Baja	Alto
06	Problemas en el hardware del equipo utilizado para el desarrollo	Baja	Alto
07	Errores en el diseño de la aplicación	Moderada	Moderado
08	Falta de potencia computacional para desarrollar y probar los modelos generados	Baja	Alto
09	Errores en la implementación de la aplicación	Baja	Alto
10	Problemas en las webs utilizadas para la obtención de datos en tiempo real	Baja	Alto

Cuadro 2.5: Tabla de riesgos del proyecto.

Id	**Plan de prevención**	**Plan de contingencia**
01	Planificación y calendarización cuidadosa y con cierta holgura.	Replanificación de las tareas e incremento del número de horas invertidas en el proyecto.
02	Mantener hábitos de vida saludable y tener precaución con la situación pandémica del COVID-19.	Aceptación y replanificación de tareas para ajustar el tiempo perdido por la indisposición.
03	Planificación y calendarización cuidadosa y con cierta holgura.	Replanificación de las tareas e incremento del número de horas invertidas en el proyecto.
04	Formación y fase de aprendizaje.	Pedir ayuda al tutor del trabajo.
05	Tener todos los archivos y programas en mi equipo personal para no perderlos en caso de fallo en la máquina virtual y para poder trabajar y probar cosas desde mi equipo.	Ponerse en contacto con los técnicos de la escuela.
06	Tener todos los archivos y programas copiados en algún dispositivo externo como un disco duro o alojados en algún servidor de nube. Hacer un uso cuidadoso del equipo utilizado para el desarrollo.	Contactar con algún amigo o familiar que pueda prestarme un equipo para continuar con el desarrollo del proyecto.
07	Realización cuidadosa y supervisada por el tutor del trabajo.	Reelaboración de las partes mal diseñadas y replanificación en caso necesario.
08	Solicitar una máquina virtual con la potencia adecuada y minimización de los recursos utilizados por el ordenador personal en el caso de ejecutarse en este la aplicación.	Solicitar una nueva máquina virtual que cumpla con los requerimientos necesarios.
09	Desarrollo con prototipos, incremental y cuidadoso.	Reelaboración de las partes mal implementadas y replanificación en caso necesario.
10	Búsqueda previa de varias fuentes de datos que proporcionen los mismos datos para tener una alternativa en caso de caída o problemas en alguna de ellas.	Uso de webs alternativas y replanificación en caso necesario.

Cuadro 2.6: Tabla de planes de prevencion y contingencia de riesgos.

Capítulo 3

Marco Conceptual

A lo largo de este capítulo se presentará el marco teórico sobre el que se desarrolla este trabajo. Se explicarán brevemente conceptos básicos sobre activos y mercados, sobre los activos elegidos en este trabajo y sobre los diferentes modelos que en él aparecen, con el fin de que todo lo que aparece en la aplicación web construida sea comprensible.

3.1 Introducción económica.

El sistema financiero de un país es el sistema constituido por entidades, mercados y medios que canalizan el ahorro generado por las unidades de gasto con superávit hacia las unidades de gasto con déficit, otorgando así seguridad al movimiento del dinero y al sistema de pagos [referencia libro de Bogle].

Algunos de los instrumentos que componen este complejo sistema son los productos bancarios (cuentas, depósitos), los planes de pensiones, los productos de seguros y los productos de inversión (acciones, bonos). Estos últimos son lo que se conoce como activos financieros [referencia del libro Mercado de valores].

Los **activos financieros** se definen como las herramientas que utiliza el sistema financiero para facilitar la movilidad de recursos. Son emitidos por una institución y adquiridos por otras instituciones o por personas particulares. Con ellos, el comprador adquiere el derecho a recibir un ingreso futuro por parte del vendedor [referencia artículo BBVA]. Los activos financieros se diferencian de los activos reales (un coche o una casa son ejemplos de activos reales) en que no suelen poseer un valor físico. Una segunda diferencia frente a estes otro tipo de activos, es que los activos económicos no incrementan la riqueza de un país (no influyen en el PIB), aunque sí contribuyen al crecimiento económico del mismo al impulsar la movilización de recursos.

Las principales características que permiten definir los distintos activos financieros son las siguientes:

- **Liquidez:** es la capacidad de transformar un activo en dinero en un espacio breve de tiempo, sin sufrir pérdidas y sin afectar a la rentabilidad. La liquidez depende tanto de la facilidad para realizar la conversión como del grado de certeza de que esa conversión se va a poder llevar a cabo sin perdidas. El activo más líquido de todos es el dinero, aunque otros activos como los fondos bancarios también gozan de gran liquidez.

- **Riesgo:** es una medida de las garantías que ofrece el vendedor, es decir, de su solvencia. Depende de la probabilidad de que este, a su vencimiento, cumpla con lo pactado con el comprador. El riesgo de un activo será mayor cuanto mayor sea la incertidumbre ofrecida por el vendedor para la devolución de la inversión. Un ejemplo de inversión de bajo riesgo sería la deuda pública

(donde el vendedor es el Estado) y ejemplos de riesgo alto podrían ser determinadas acciones de bolsa o inversiones en ciertas criptomonedas.

- **Rentabilidad:** se define como el interés o la plusvalía que obtiene el comprador a cambio de aceptar el riesgo de ceder su capital. Puede reflejarse en términos de intereses, beneficios u otros. Por lo general, cuanto mayor es la rentabilidad, mayor es el riesgo asociado al activo y menor es su liquidez. Por el contrario, cuanto menor es el riesgo que ofrece un emisor y mayor es la liquidez, menor es también el beneficio que se espera obtener.

Otro aspecto importante a tratar a la hora de definir los activos financieros es su clasificación. Usualmente esta se divide en dos categorías:

- **Activos primitivos:** aquellos cuyo valor depende de las rentas futuras esperadas y de su riesgo asociado. A su vez se dividen en otras dos categorías:
  - **Renta fija:** emitidos por entidades públicas o empresas, que previamente fijan el pago de un interés (de una cantidad y en un plazo establecido), de forma que este no depende de la evolución ni de los resultados de la compaía emisora, quedando este siempre garantizado. Algunos ejemplos de este tipo de activos son los bonos de deuda emitidos por los Estados o los pagarés emitidos por las empresas.
  - **Renta variable:** se describen así los activos cuyo rendimiento es variable en función de la marcha de la entidad emisora. En este caso, ni la rentabilidad ni la recuperación del capital están garantizados. El ejemplo más importante de estos activos financieros son las acciones.
- **Activos derivados:** reciben este nombre porque su valor depende del valor de un activo primitivo denominado *activo subyacente*. El activo derivado es la *opción* que depende del activo primitivo, como, por ejemplo, una opción sobre una acción. Este tipo de activos permiten a las empresas protegerse frente a fluctuaciones. Tipos de activos derivados son:
  - **Contratos a plazo *forward*:** contratos de compraventa negociados directamente entre comprador y vendedor (sin un mercado organizado como, por ejemplo, el mercado de valores) en los que se establece el precio a pagar, en una fecha futura, ante la entrega de un activo.
  - **Contratos futuros:** contrato que obliga a las partes a la compraventa de bienes o activos con una fecha y precio previamente establecidos. Se diferencian de los contratos *forward* en que se negocian en un mercado organizado que hace las veces de intermediario y en que, en general, pueden transmitirse a terceros.
  - **Opción:** derecho a vender un activo (el activo subyacente) en una fecha futura, a un precio fijado. Se distingue entre opciones de compra (*call*) y opciones de venta (*put*).
  - ***Swaps:*** acuerdos para el intercambio futuro de flujos monetarios según unas reglas previamente determinadas.

Una última clasificación sobre activos financieros que es importante comentar, es aquella determinada por los plazos de vencimiento. Distinguiéndose de este modo entre:

- **Activos a corto plazo:** se amortizan en un plazo corto de tiempo, generalmente inferior a 12 meses. Suelen ofrecer rentabilidades más bajas y algunos ejemplos son los pagarés y las letras del tesoro.
- **Activos a medio y largo plazo:** su amortización es en un plazo superior a 12 meses. Se negocian en los mercados de capitales y su riesgo es mayor (y, por tanto, mayor también su rentabilidad en términos generales) debido a la mayor fluctuación y volatilidad que puede darse en los valores y en los tipos de interés al ampliarse el plazo temporal. Algunos ejemplos de estos activos son las acciones o los bonos.

Finalmente, comentar que existen otras maneras de clasificar los activos financieros, como la liquidez, la valoración, el tipo de entidad emisora, el plazo de vencimiento o las características legales de la relación vendedor-comprador, aunque no se comentará más acerca de las mismas.

3.2 Bolsa.

3.2.1 Fundamentos y conceptos.

3.2.2 S&P 500.

3.2.3 NASDAQ.

3.2.4 IBEX35.

3.3 Criptodivisas.

3.3.1 Fundamentos y conceptos.

3.3.2 Blockchain.

3.3.3 Bitcoin.

3.3.4 Otras criptodivisas.

3.4 Materias primas.

3.4.1 Oro.

3.4.2 Otras materias primas.

3.5 Divisas.

3.6 Análisis de datos de activos.

3.6.1 Análisis fundamental.

3.6.2 Análisis técnico.

3.7 Optimización de carteras.

3.7.1 Modelo básico.

3.7.2 Modelo de Markowitz.

3.7.3 Modificaciones al modelo de Markowitz.

Capítulo 4

Soluciones Existentes

Capítulo 5

Análisis

Capítulo 6

Diseño

6.1 Diseño

Capítulo 7

Implementación

7.1 Despliegue del servidor

El objetivo de este trabajo desde un primer momento fue la creación de una aplicación web (como se detalló en el apartado 1.3). Como se pretendió desde un primer momento poder acceder a ella desde cualquier dispositivo y lugar a través de internet, se decidió que lo mejor era desplegar el servicio web en una máquina virtual de la escuela.

La máquina solicitada fue una ubuntu 20.04.3 LTS de 2 núcleos con memoria RAM de 8G y memoria de disco de 16G.

7.1.1 Docker

El sistema elegido para realizar el montaje del servidor web en la máquina virtual fue Docker [referencia a la web de Docker]. Esta elección se debió a la facilidad que provee este sistema para crear, probar e implementar aplicaciones de un modo rápido. Docker permite llevar a cabo la virtualización de un sistema operativo, pudiéndose desplegar y manipular múltiples sistemas en una misma máquina. [referencia de wikipedia] Docker proporciona un nivel extra de abstracción y utiliza características de aislamiento de Linux como los grupos y los espacios de nombres para permitir que los distintos contenedores se ejecuten dentro de una única instancia de Linux. Estos contendores son unidades que incluyen todo lo necesario para que un determinado software se ejecute (librerías, herramientas de sistema, código, etc) y constituyen el punto clave de Docker. Su propósito es ejecutar varios procesos de manera separada, de forma que todos mantengan las características de seguridad que tendrían ejecutándose como sistemas individuales y permitiendo así un mayor aprovechamiento de la infraestructura.
Los sistemas Docker se componen de dos elementos, las imágenes y los contenedores:

- **Imágenes:** una imagen es la definición de un sistema operativo. Solamente ha de instalarse una vez y compartirla permite replicar el sistema Docker construido.
- **Contenedores:** son instancias de la imagen y han de ser cargados cada vez que requiramos una instancia. Se pueden ejecutar al mismo tiempo varios contendores de una misma imagen y contienen todo lo necesario para que las aplicaciones puedan ejecutarse.

Ventajas de los contendores Docker[referencia a dockerfile reference]:

- **Modularidad:** la separación en contenedores permite modificar una parte de una aplicación sin que el resto se vea alterado, pudiendo así actualizarla, repararla o ampliarla.

- **Control de versiones de imágenes y restauración:** Docker está basado en imágenes, lo que permite compartir un sistema o aplicación con todos sus elementos en cualquier entorno. A su vez, una imagen Docker está formada por varias capas de manera que al ejecutar un comando, la imagen se modifica con la creación de una nueva capa. Esto permite controlar los cambios realizados mediante el registro de los mismos en las imágenes y poder volver a versiones anteriores de un sistema en caso de requerirse.
- **Rapidez:** Docker permite establecer un servicio en pocos segundos. Además, cada proceso se encuentra en un contendor distinto, lo que permite compartirlos con aplicaciones o sistemas nuevos. A su vez, como los cambios pueden realizarse a nivel de capa o contenedor, no es necesario volver a reiniciar o cargar el sistema operativo.

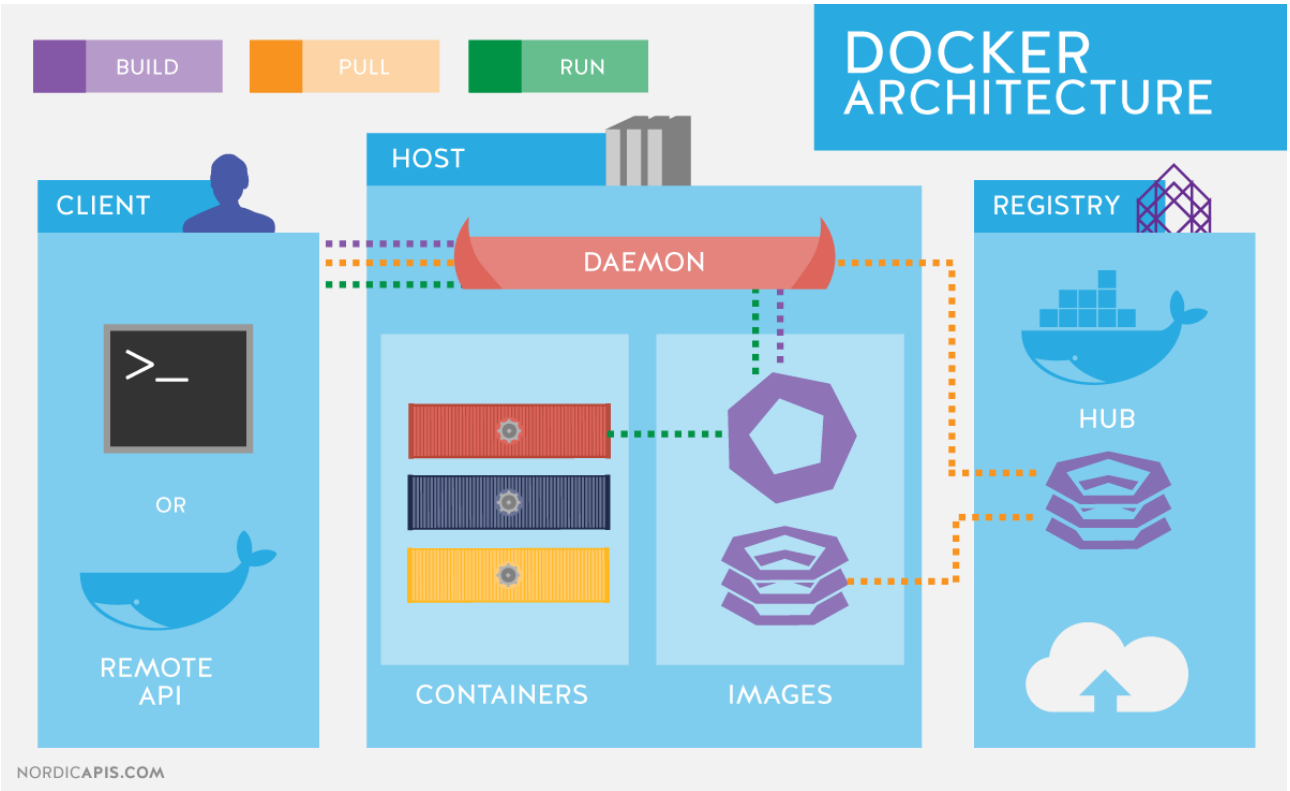


Figura 7.1: Arquitectura Docker.

Toda aplicación Docker tiene un archivo de configuración denominado ”dockerfile”que contiene las instrucciones para construir una imagen. Este archivo puede constar de varias instrucciones, entre las que cabe destacar las siguientes[referencia]:

- **FROM:** describe qué imagen se está construyendo.
- **RUN:** comandos que imitan la línea de comandos.
- **COPY:** ficheros con el código que se desea ejecutar. Han de estar en el mismo directorio que el dockerfile.
- **CMD:** comando a ejecutar cada vez que es lanza el Docker.
- **WORKDIR:** establece el directorio de trabajo.

Otra instrucción interesante es ARG WHEN, que permite especificar la fecha que queremos que se use en los paquetes y librerías de la aplicación, independientemente de cuándo se utilice esta.

En el apéndice A se incluye la guía que describe los pasos seguidos en este trabajo con la utilización de docker para el despliegue del sistema.

7.2 Herramientas de Desarrollo

7.3 Implementación

Capítulo 8

Pruebas

Capítulo 9

Conclusiones

9.1 Aportaciones

9.2 Trabajo futuro

Appendices

Apéndice A

Manual de Instalación

Escribir introducción...

A.1 Despliegue servidor web

Tras haber comentado los aspectos principales de Docker y de cómo se utilizan este tipo de sistemas en la sección 7.1, se proporciona un manual de instalación completo a fin de que pueda replicarse el sistema utilizado en el presente trabajo.

A.1.1 Pasos llevados a cabo para el despliegue del servidor.

Si se desea instalar la imagen utilizada en este proyecto, los pasos a seguir son los siguientes:
Aquí comentar el tema de descargar de mi github la imagen y cómo instalarla, rellenar esta sección cuando la imagen esté creada, aunque no sea aún la imagen final del sistema.
Si, por el contrario, se desean replicar los pasos llevados a cabo en este trabajo para la creación desde 0 del servidor Docker, los pasos a seguir en una consola de comandos de Linux entonces son los siguientes.
(**Nota:** replicando estos pasos en nuestro equipo personal puede ser necesario tener que dar permisos o tener que ejecutar los comandos como superusuario con `sudo`.):

1. **Descarga de Docker:** `sudo apt install docker.io`

   Con el comando `docker system info` se pueden ver las características del sistema Docker instalado, que en este caso son las mostradas en la Figura A.1.

   Del mismo modo, la información de las imágenes instaladas puede verse con el comando

   `docker image ls`

2. Modificación del fichero `/etc/group/` para incluir un nuevo usuario. Esto se puede hacer con vi, vim o cualquier otro editor de texto.

3. **Explicacion de como crear la imagen que necesitamos para la web...**

A.1.2 Conexión al servidor a través de Visual Studio Code.

Visual Studio Code [Referencia a la web de vsc] es uno de los IDE (*Integration Development Environment*) más populares a día de hoy. Desarrollado por Microsoft para Windows, Linux y macOS, de uso gratuito y de código abierto, debe su fama a la gran cantidad de opciones que proporciona para el

desarrollador, entre las que cabe destacar: soporte para depuración, integración de control de versiones con git, gran cantidad de lenguajes con herramientas de ayuda de sintaxis, finalización inteligente de código y refactorización, así como múlitples opciones de personalización del entorno.

Visual Studio Code permite a su vez trabajar de manera remota con un servidor, teniendo en todo momento el código alojado en este. Esta característica la aprovechamos en el presente trabajo utilizando una conexión **ssh** con la máquina virtual utilizada en el proyecto. Hay dos posibilidades, a través del usuario y contraseña del servidor, o utilizando claves públicas y privadas.

Figura A.1: Logotipo de Visual Studio Code.

- **Con usuario y contraseña.**

  1. Instalación de Visual Studio Code en la máquina desde la que se va a trabajar.
  2. Instalación de la extensión *Remote-ssh* desde el menú de extensiones de Visual Studio Code.
  3. Presionar la tecla F1 y escribir *remote* en el buscador que se nos abre.
  4. Hacer click en la opción *Remote-SSH: Add New Host...*.
  5. Introducimos la dirección ssh del servidor al que nos queremos conectar, en este caso, la dirección ssh de la máquina virtual: *ssh victor@virtual.lab.inf.uva.es*.
  6. Escogemos la dirección en donde se guardarán las credenciales SSH para la conexión remota con nuestro servidor (normalmente, la primera opción que nos aparece).
  7. En caso de desearlo podemos editar el fichero *config* seleccionando *Open config* en el pop up que se nos muestra en la parte inferior derecha de la pantalla. Aquí podemos cambiar el nombre del host para facilicar la identificación del mismo.
  8. Seleccionamos la opción connect de dicho pop up e introducimos la contraseña de nuestro usuario en el servidor y la conexión quedará establecida. Los archivos del servidor son accesibles a través del explorador de archivos.
  9. Para realizar conexiones a dicho servidor en nuevas sesiones tendremos que dar nuevamente F1 y seleccionar la opción *Remote-SSH: Connect Current Window to Host...* y seleccionar la conexión que queremos establecer.
  10. Para desconectarse del servidor pinchamos en la parte inferior izquierda de la pantalla en el recuadro donde pone *SSH: nombre_conexion* y seleccionamos *Cerrar conexión remota* en las opciones que se nos muestran.

- **Con claves públicas y privadas.**

  1. Seguimos los dos primeros pasos del procedimiento anterior. Tras ello, procedemos a la generación de claves en nuestra máquina local. En un terminal, escribimos el comando `sshkeygen` para la generación de las claves. Esto genera dos archivos (par de claves RSA): *id_rsa* e *id_rsa.pub* donde el segundo de ellos es la clave pública que debemos copiar a nuestro servidor.
  2. Para copiar la clave pública utilizamos el siguiente comando en el terminal: `sshcopyid i $HOME/.ssh/id_rsa.pub usuario@servidor`.
  3. La clave pública se habrá copiado en el servidor en el directorio oculto *$HOME/.ssh* en un archivo llamado *authorized_keys*.
  4. El modo de proceder desde Visual Studio Code una vez establecida la conexión con el servidor es análogo al descrito en el procedimiento anterior, con la salvedad de que ahora editamos el fichero *config* añadiendo un campo *IdentifyFile* seguido de la ruta al directorio donde esté nuestra clave privada (*$HOME/.ssh/clave_privada*).

5. Si al establecer la conexión salta algún cartel indicando que no se ha podido establecer la conexión, tendremos que dar permisos de lectura al fichero de la clave privada, con el comando `sudo chmod 400 clave_privada`.

Apéndice B

Manual de Usuario

Apéndice C

Manual del Desarrollador

Revisar que las referencias WEB contengan fecha de último acceso

Centralizar items de bibliografía en el .bib

Bibliografía

[1] J.M.G. García y col. *Cómo escribir un trabajo de fin de grado: algunas experiencias y consejos prácticos*. Colección Síntesis. Editorial Síntesis, 2014. ISBN: 9788490770481. URL: https://books.google.es/books?id=xpcWogEACAAJ.