



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



Escola Tècnica
Superior d'Enginyeria
Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Implementación de un sitio web para un concurso de programación paralela

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Nicolás Fernando Martini

Tutor: Pedro Alonso Jordá

Curso 2019-2020

Dedicatoria

...

Agradecimientos

A mi tutor Pedro Alonso Jordá por su apoyo y la oportunidad de hacer este proyecto con la esperanza que produzca un impacto positivo en la enseñanza en los cursos venideros.

A la ETSINF y todos sus profesores, gracias a ellos he podido crecer personal y profesionalmente.

Al Ministerio de Educación y Gobierno de España, gracias a sus ayudas he podido acceder los estudios de grado.

Resum

...
...
...

Paraules clau: ?, ?, ?, ?

Resumen

El presente trabajo aborda la implementación de un sitio web para concursos de programación paralela, con el añadido de la gamificación o ludificación, una técnica de aprendizaje que busca recompensar al usuario y aumentar su motivación al sumar elementos y dinámicas propias de los juegos para así ofrecer una experiencia enriquecedora y positiva.

Esta es una tarea compleja, por un lado llevar el proceso de envío de código a un entorno web y la interacción que tendrá con el cluster kahan del DSIC. Y por el otro, transformar las actividades de laboratorio de las asignaturas CPA y LPP con nuevas mecánicas que produzcan al estudiante buscar mejorar sus resultados inclusive después de llegar a una resolución correcta a los ejercicios planteados.

Este proyecto ha sido realizado con el apoyo del *DSIC* de la *UPV* con la finalidad de complementar otras herramientas utilizadas hoy en día en la enseñanza.

Palabras clave: programación paralela, concurso de programación, gamificación, ludificación

Abstract

...
...
...

Key words: parallel programming, programming competition, gamification

Índice general

Índice general	VII
Índice de figuras	IX
Índice de tablas	IX
<hr/>	
1 Introducción	1
1.1 Motivación	1
1.2 Objetivo	1
1.3 Estructura de la memoria	2
2 Estado del arte	3
2.1 Crítica al estado del arte	3
2.2 Propuesta	3
3 Análisis del problema	5
3.1 Actores	5
3.2 Historias de usuario	5
3.3 Requisitos funcionales	6
3.4 Requisitos no funcionales	9
4 Diseño de la solución	11
4.1 Software	11
4.1.1 Paquetes de Python	12
4.2 Diseño de la Aplicación en Flask	14
4.3 Modelo de datos	15
4.4 Envío de soluciones de Tareas	17
4.5 Sistema de colas	18
4.6 Arquitectura de la solución	18
5 Desarrollo de la solución	21
5.1 Entregables	21
6 Implantación	23
6.1 Docker y Kubernetes	23
6.2 Despliegue en la Nube con Kubernetes	25
6.2.1 Pasos de un despliegue en GCE	25
6.2.2 Costos asociados	27
7 Mantenimiento	29
7.1 Entorno local de desarrollo	29
7.2 Depuración de errores	31
7.3 Calidad de código	31
8 Extensibilidad	33
8.1 Otros sistemas de Gestión de Tareas	33
8.2 Localización	34
9 Conclusiones	37
9.1 Relación del trabajo desarrollado con los estudios cursados	37
10 Trabajos futuros	39

Bibliografía	41
<hr/>	
Apéndices	
A Configuración del sistema	43
A.1 Inicialización	43
A.2 Parámetros	44
B Licencia	47

Índice de figuras

4.1	AdminLTE: plantilla de Panel de Control	12
4.2	Diagrama UML de la Base de Datos	16
4.3	Diagrama de flujo de una petición web para un Envío	19
4.4	Proceso que sigue un Envío en la Cola interna	20
6.1	Diferencias entre contenedor Docker y máquina virtual	23
6.2	nuevo proyecto en GCE	25
6.3	Calculadora de precios en GCE	28
6.4	Estimación de costos en GCE	28
7.1	Inicio de sesión en Pizarra	30
8.1	Selector de Idiomas en Pizarra	35

Índice de tablas

3.1	RF-1 Registro de usuarios	6
3.2	RF-2 Login de usuarios	6
3.3	RF-3 Equipos	7
3.4	RF-4 Dashboard	7
3.5	RF-5 Mis Envíos	7
3.6	RF-6 Envío	8
3.7	RF-7 Visualización de Tareas	8
3.8	RF-8 Visualización de Tarea	8
3.9	RF-9 FAQ	8
3.10	RF-10 Tabla de Posiciones	9
3.11	RF-11 Perfil	9
3.12	RNF-1 Usabilidad	9
3.13	RNF-2 Implantación	10
3.14	RNF-3 Configuración	10
3.15	RNF-4 Localización	10

CAPÍTULO 1

Introducción

En este proyecto se trabajarán dos conceptos importantes, la programación paralela y la gamificación para llevarlos a un terreno en conjunto y así crear una herramienta que motive a los estudiantes en el aprendizaje de la asignatura de *LPP*.

La programación paralela es una rama importante de la computación donde se buscar partir problemas de gran magnitud en pedazos más pequeños donde cada partición es ejecutada de forma simultanea por ...

La gamificación como herramienta para motivar

1.1 Motivación

Desde el inicio de mis estudios en el Grado de Ingeniería Informática me han interesado los concursos de programación como los promovidos desde la propia *ETSINF* así como también los disponibles en plataformas online como UVa¹, HackerRank², CheckIO³ y Project Euler⁴. Estas competiciones ayudan a promover el interés en diferentes ramas de los estudios cursados con el añadido de un marco competitivo.

Normalmente, los concursos de programación abarcan problemas relacionados con algoritmia, y al ver la publicación de la propuesta del *TFG* donde se añade la programación paralela me ha despertado el interés ya que no solo se busca que se encuentre la solución al problema planteado, sino que también el punto más importante es la eficiencia.

Todo esto, además de poder contribuir en crear una herramienta que utilizarán los alumnos de la universidad me ha motivado a realizar esta aplicación que expongo en la memoria.

1.2 Objetivo

El principal objetivo es crear una herramienta

¹UVa Online Judge website: <https://onlinejudge.org/>

²HackerRank website: <https://www.hackerrank.com/>

³CheckIO website: <https://checkio.org/>

⁴Project Euler website: <https://projecteuler.net/>

1.3 Estructura de la memoria

La memoria está dividida en los siguientes capítulos:

- **Estado del arte:** se analizan las alternativas que existen al trabajo planteado.
- **Análisis del problema:** se recopila los requerimientos del proyecto para así llegar a una propuesta que abarque las necesidades actuales de la asignatura *LPP*.
- **Diseño de la solución:** se describe todos los elementos a nivel de arquitectura que forman parte de la solución.
- **Desarrollo de la solución:** ?
- **Implantación:** se explica los pasos a seguir para desplegar el software.
- **Pruebas:** se describe las pruebas que se han realizado para verificar el correcto funcionamiento.
- **Mantenimiento:** ?
- **Conclusiones:** ?

CAPÍTULO 2

Estado del arte

2.1 Crítica al estado del arte

2.2 Propuesta

CAPÍTULO 3

Análisis del problema

Antes de comenzar a escribir una línea de código, como todo proyecto ingenieril se debe hacer un estudio que involucre los usuarios que utilizarán el sistema donde se recogerán requisitos para poder hacer una propuesta acorde a sus necesidades.

3.1 Actores

El sistema contará con dos perfiles de usuarios que llamaremos actores, estos son el *Estudiante* y *Administrador*. También, veremos que en la memoria se hace mención al *usuario*, esto se hace para referirnos a funcionalidades que se aplican tanto a los *Estudiantes* como *Administradores*.

- **Estudiante:** el rol principal del sistema. Enviará soluciones a los problemas propuestos y tratará de mejorar sus resultados para subir su ranking en la tabla de posiciones de las tareas y la grupal. Su trabajo será recompensado en forma de puntos e insignias.
- **Administrador:** el rol que se encarga de poner el sistema en marcha. Tendrá que tener conocimientos sólidos en la asignatura de *CPA* para poder crear tareas que varíen en dificultad y motive al alumnado a mejorar sus resultados.

3.2 Historias de usuario

Como primer paso se describe el resultado esperado de este proyecto en un lenguaje sencillo que más adelante darán partida a los requisitos funcionales y no funcionales de la aplicación. A esto se lo llaman **Historias de usuario**.

- Como *Estudiante*, quiero ingresar al sistema utilizando mis credenciales personales.
- Como *Estudiante*, quiero poder cambiar mi contraseña.
- Como *Estudiante*, quiero registrarme en el sistema para poder hacer uso del mismo.
- Como *Estudiante*, quiero enviar las resoluciones a mis tareas asignadas.
- Como *Estudiante*, quiero formar parte de un equipo con otros estudiantes.
- Como *Estudiante*, quiero acceder a mi perfil para ver toda mi información.

- Como *Estudiante*, quiero visualizar los resultados de las ejecuciones para ver como han quedado posicionadas en la tabla de clasificación.
- Como *Estudiante*, quiero ver un resumen de mi actividad.
- Como *Administrador*, quiero crear grupos para que los estudiantes puedan formar parte de ellos.
- Como *Administrador*, quiero crear estudiantes subiendo un fichero csv o txt.
- Como *Administrador*, quiero crear y asignar tareas a diferentes grupos.
- Como *Administrador*, quiero crear insignias para asignar a diferentes tareas.
- Como *Administrador*, quiero editar y eliminar grupos, estudiantes, tareas e insignias.
- Como *Administrador*, quiero desplegar el aplicativo en una solución cloud.

3.3 Requisitos funcionales

En el siguiente apartado se describen todos los requisitos funcionales del aplicativo.
... explicación de la tabla ...

RF-1

Nombre	Registro de usuarios
Descripción	El sistema permitirá el registro de usuarios mediante dirección de correo electrónico
Prioridad	Media
Criterio de aceptación	<ul style="list-style-type: none"> no puede existir más de un usuario con el mismo correo electrónico se generará un usuario único con el alias del correo electrónico se podrá deshabilitar el registro de usuarios mediante configuración del sistema se podrá limitar el registro de usuarios a correos electrónicos que pertenezcan a dominios específicos (eg: @upv.es, @inf.upv.es)

Tabla 3.1: RF-1 Registro de usuarios

RF-2

Nombre	Login de usuarios
Descripción	El sistema permitirá el login de usuarios mediante dirección de correo electrónico o usuario
Prioridad	Más Alta
Criterio de aceptación	<ul style="list-style-type: none"> solo se podrá ingresar al sistema si la cuenta está activa

Tabla 3.2: RF-2 Login de usuarios

RF-3

Nombre	Equipos
Descripción	Los <i>Estudiantes</i> podrán formar parte de un equipo con otros compañeros de grupo
Prioridad	Baja
Criterio de aceptación	<ul style="list-style-type: none"> • solo se podrá formar parte de un único equipo en un momento determinado de tiempo • si un equipo no tiene integrantes se eliminará del sistema • los equipos contarán con una URL única que permitirá a otros estudiantes unirse a los mismos • un <i>Estudiante</i> no puede unirse un equipo que no forme parte de su grupo o que haya llegado al máximo de integrantes • los equipos tendrán un máximo de integrantes que podrá ser configurado por el <i>Administrador</i>

Tabla 3.3: RF-3 Equipos**RF-4**

Nombre	Dashboard
Descripción	Los <i>Estudiantes</i> al ingresar verán un resumen agregado de su estado general en el sistema
Prioridad	Alta
Criterio de aceptación	<p>El Dashboard o <i>Página de Inicio</i> del <i>Estudiante</i> deberá mostrar la siguiente información:</p> <ul style="list-style-type: none"> • resumen de los últimos envíos • extracto calculado de cantidad de envíos, tareas, cuota disponible y puntaje • la última insignia obtenida, en caso que no tuviere, alentarlos a completar una tarea para conseguir su primera • los integrantes del equipo, en caso que no tuviere, alentarlos a crear uno nuevo o unirse a uno existente

Tabla 3.4: RF-4 Dashboard**RF-5**

Nombre	Mis Envíos
Descripción	Los <i>Usuarios</i> podrán ver un resumen de sus envíos enviados al sistema
Prioridad	Muy Alta
Criterio de aceptación	<p>El resumen de envíos se visualizará en forma de tabla, se podrá filtrar por la información disponible y deberá mostrar las siguientes columnas:</p> <ul style="list-style-type: none"> • ID único en el sistema • Fecha y Hora de envío • Tarea a la que corresponde • Estado del envío • Puntaje obtenido • Tiempo de ejecución

Tabla 3.5: RF-5 Mis Envíos

RF-6

Nombre	Envío
Descripción	Visualización del envío a ejecutar
Prioridad	Alta
Criterio de aceptación	Se deben mostrar los siguientes apartados e información: <ul style="list-style-type: none"> • código fuente enviado • análisis estático del código fuente enviado • output del código ejecutado • tiempo de ejecución y status

Tabla 3.6: RF-6 Envío**RF-7**

Nombre	Tareas
Descripción	Visualización de listado de Tareas
Prioridad	Alta
Criterio de aceptación	Se deben mostrar los siguientes apartados e información: <ul style="list-style-type: none"> • si el usuario es un <i>Estudiante</i> se deben mostrar las tareas asignadas a su grupo • si el usuario es un <i>Administrador</i> se deben mostrar todas las tareas en el sistema • por cada Tarea debe haber un resumen, insignias y últimos envíos • no se deben mostrar las insignias secretas

Tabla 3.7: RF-7 Visualización de Tareas**RF-8**

Nombre	Tarea
Descripción	Visualización de Tarea
Prioridad	Alta
Criterio de aceptación	Se deben mostrar los siguientes apartados e información: <ul style="list-style-type: none"> • descripción completa • insignias a obtener • insignias secretas que el usuario ha obtenido • status de la tarea: abre pronto, abierto, cierra pronto, cerrada • si el usuario ha enviado una ejecución satisfactoria se le debe informar en un mensaje • enlace para hacer un envío con sus credenciales • enlace a la tabla de posiciones

Tabla 3.8: RF-8 Visualización de Tarea**RF-9**

Nombre	FAQ
Descripción	Página con preguntas frecuentes que se pueden hacer los usuarios del sistema y sus respuestas
Prioridad	Baja
Criterio de aceptación	<ul style="list-style-type: none"> • debe estar localizado en al menos 2 idiomas • no debe contener más de 10 preguntas

Tabla 3.9: RF-9 FAQ

RF-10

Nombre	Tabla de Posiciones
Descripción	
Prioridad	Alta
Criterio de aceptación	• ?

Tabla 3.10: RF-10 Tabla de Posiciones**RF-11**

Nombre	Perfil
Descripción	
Prioridad	Media
Criterio de aceptación	• ?

Tabla 3.11: RF-11 Perfil

3.4 Requisitos no funcionales

Los requisitos no funcionales son criterios que se deben cumplir para juzgar la correcta operación del sistema. En contraste con los requisitos funcionales, no definen comportamientos específicos.

... explicación de la tabla ...

RNF-1

Nombre	Usabilidad
Descripción	El sitio web deberá tener una interfaz sencilla y fácil de utilizar.
Prioridad	Muy Alta

Tabla 3.12: RNF-1 Usabilidad

RNF-2

Nombre	Implantación
Descripción	El aplicativo debe poder implantarse de forma automatizada
Prioridad	Media

Tabla 3.13: RNF-2 Implantación**RNF-3**

Nombre	Configuración
Descripción	El aplicativo debe permitir configurar los parámetros de despliegue y uso
Prioridad	Alta

Tabla 3.14: RNF-3 Configuración**RNF-4**

Nombre	Localización
Descripción	El aplicativo debe soportar la localización en diferentes idiomas
Prioridad	Media

Tabla 3.15: RNF-4 Localización

CAPÍTULO 4

Diseño de la solución

Una vez analizados los requisitos funcionales y no funcionales el próximo paso es hacer el diseño del software a implementar. Aunque la finalidad de este trabajo es la creación de un sitio web, existen elementos adicionales que deben considerarse para cumplir con el objetivo. El punto de entrada de interacción del sistema será la aplicación web que deberá persistir los datos en una BD, como también los envíos de resolución de Tareas en un sistema de ficheros compartido NFS y deberá soportar s locales como también poder conectarse al cluster kahan.

Todos estos puntos son los pilares en los cuales debe fundarse la aplicación y en este apartado se encuentra el proceso de análisis para la elección del software, sus componentes y el diseño de la arquitectura del sistema.

4.1 Software

Muchos de los requisitos para la aplicación web ya se encuentran implementados de forma similar en *Tablón*, y por esto son un excelente punto de partida para nuestro trabajo. Como *Tablón* está desarrollado en Python, siendo un lenguaje de programación fácil de aprender, con una extensa cantidad de Paquetes y guías en Internet, se ha decidido hacer el desarrollo en éste lenguaje.

Además, dadas ciertas limitaciones que tiene *Tablón* una de las aspiraciones en la creación de ésta aplicación web para concursos de programación paralela es que sea su sucesor (o reemplazo) a futuro. Por esto mismo he decidido llamar a la aplicación web *Pizarra* en honor a *Tablón*. De aquí en adelante, cuando se nombre *Pizarra* se hace referencia a la aplicación web y otros elementos adicionales necesarios para su funcionamiento.

Ya escogido el lenguaje de programación, el próximo paso es analizar las posibilidades para la aplicación web en sí que será el punto de acceso a los usuarios. Debido a experiencia previa en otra asignatura de la carrera universitaria he decidido optar por *Flask*¹, un micro-framework de Python con gran acogida por la comunidad que incluye un sistema de plantillas para la generación de páginas web llamado Jinja.

La usabilidad es uno de los aspectos más importantes de una aplicación web y en este caso no se puede dejar de lado. Un usuario de *Pizarra* esperará una interfaz intuitiva y fácil de utilizar que le facilite el proceso de envío de soluciones a los problemas propuestos; No queremos que la aplicación se interponga entre el proceso de aprendizaje y la satisfacción de participar y obtener logros. Una gran cantidad de aplicaciones utilizan las mismas librerías para maquetar los sitios web, entre estas librerías la más importante es

¹Sitio web Flask: <https://flask.palletsprojects.com/en/1.1.x/>

*Bootstrap*², y además, a partir de estas librerías existen Dashboards o Paneles de Control de código abierto para utilizar como punto de partida en nuevas aplicaciones.

Después de varias búsquedas y pruebas se ha optado por *AdminLTE*³ dada su variedad de plantillas disponibles, uso de otras librerías con amplia documentación y un aspecto visual amigable y moderno.

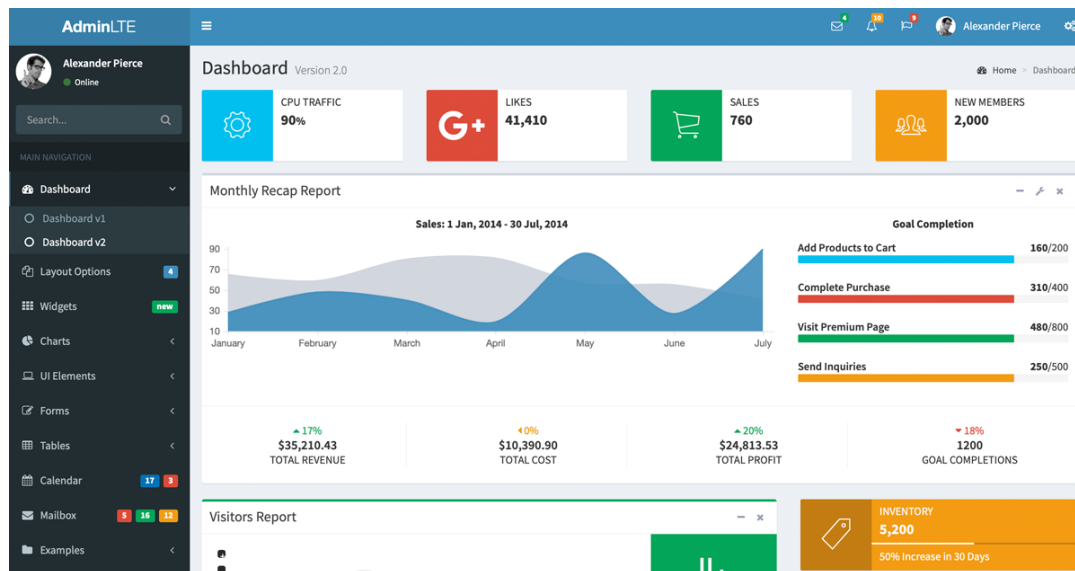


Figura 4.1: AdminLTE: plantilla de Panel de Control

4.1.1. Paquetes de Python

Python será el lenguaje a utilizar, pero para dar funcionalidades adicionales normalmente se utilizan librerías o Paquetes externos que simplifican y aceleran el desarrollo de la solución. A continuación se hace un repaso de las librerías que son necesarias para *Pizarra*, este listado se encuentra disponible en el fichero **requirements.txt**, un estándar utilizado para su fácil instalación a posteriori por administradores de paquetes como *pip*.

Flask

El framework base de la aplicación, con otros Paquetes para dar soporte a ciertas funcionalidades no incluidas en la librería base.

- flask: framework
- flask_login: funcionalidades extras para el manejo de sesiones de usuario
- flask_security: uso de roles en el sistema
- flask_wtf: validación de formularios web
- flask_sqlalchemy: soporte a SQLAlchemy para el mapeo objeto-relacional a diferentes BD
- flask_migrate: migración de base de datos basadas en SQLAlchemy

²Sitio web Bootstrap: <https://getbootstrap.com/>

³Sitio web AdminLTE: <https://adminlte.io/>

- flask-babel: localización de la aplicación en diferentes idiomas
- email-validator: validación de correos electrónicos para la librería wtf

Colas

Existen varios sistemas de Colas en Python, en este caso se ha optado por uno que utiliza pocos recursos y permite encolar tareas a ejecutar en cierto determinado de tiempo en el futuro, funcionalidad que será útil para revisar los resultados de ejecuciones en kahan. *RQ*⁴ (Redis Queue) permitirá desacoplar las colas del aplicativo web al utilizar *Redis* además de disponer de un dashboard que puede ser embebido en cualquier aplicación web.

- rq: librería de Redis Queue
- rq-dashboard: visualización web del estado de Colas y Tareas
- redis: soporte y conectividad a Redis

Datos de ejemplo

Para facilitar el desarrollo y su posterior mantenimiento se debe poder disponer de una forma fácil y estandarizada para carga de datos de ejemplo. Para esta funcionalidad se ha optado por ficheros *yaml*.

- pyyaml: analizador de ficheros *yaml*
- sqlalchemy-fixtures: definición de datos via *yaml* para su posterior importación a SQLAlchemy

Conexión con sistemas externos

Para poder utilizar las Colas de kahan es necesario poder conectarnos vía protocolos seguros ya sea para ejecutar comandos o copiar ficheros.

- paramiko: soporte de protocolo SSH
- scp: soporte de protocolo SCP

Otros

- gunicorn: ejecución y monitoreo de la aplicación
- lizard: analizador de código estático
- rule_engine: evaluador de reglas con expresiones lógicas, utilizado para asignar Insignias

⁴Sitio web RQ: <https://python-rq.org/>

4.2 Diseño de la Aplicación en Flask

Al crear una nueva aplicación en *Flask* existen ciertos patrones o mejores prácticas a seguir, entre ellas se encuentra uno de los patrones de diseño más comunes en la ingeniería de software, Modelo Vista Controlador (MVC). Con este patrón se crea una separación de capas entre lo que es el Modelo que contiene la representación de datos y persistencia, la Vista que es la interfaz de usuario y el Controlador como intermediario entre el Modelo y la Vista.

Una aplicación estándar en *Flask* incluye los siguientes ficheros:

- `models.py`: representación del modelo de datos (Modelo)
- `templates`: carpeta con las plantillas HTML para la generación de páginas web (Vista)
- `routes.py`: rutas de acceso a la aplicación (Controlador)
- `__init__.py`: inicialización de la aplicación y librerías
- `forms.py`: formularios y validación

Además, otra buena práctica es separar las aplicaciones en módulos para añadir extensibilidad mediante reglas, nuevas vistas, separación de roles y otros beneficios añadidos mediante el uso de blueprints.

De esta forma se ha optado por crear los siguientes módulos o blueprints, todos estos dentro del directorio **app**:

- `account`: apartado de “mi cuenta” de un usuario
- `admin`: panel de control del administrador, solo accesible para usuarios con ese rol
- `base`: contiene la definición de Modelos y plantillas base.
- `data`: importación de datos de ejemplo
- `home`: dashboard de usuario

Si se lista el directorio **app** se puede ver como cada módulo contiene una estructura de datos similar, siguiendo el estándar descrito previamente.

```
$ tree -L 2 app/
app/
|-- __init__.py
|-- account
|   |-- __init__.py
|   |-- forms.py
|   |-- routes.py
|   +-- templates
|-- admin
|   |-- __init__.py
|   |-- forms.py
|   |-- routes.py
|   +-- templates
|-- base
```



```

|--- __init__.py
|--- forms.py
|--- models.py
...

```

4.3 Modelo de datos

Uno de los Paquetes más importantes utilizados en la aplicación es *SQLAlchemy* que permite mapear los objetos de Python a una base de datos de tipo relacional. *SQLAlchemy* soporta varios motores de base de datos, entre otros SQLite, MySQL, PostgreSQL, Oracle o MS SQL. En este caso se optará por SQLite para el entorno de desarrollo local (se crea el fichero *database.db* en el directorio raíz del aplicativo) y PostgreSQL para un entorno productivo dado que son de libre uso.

Dados los requerimientos funcionales, se han identificado y creado las siguientes clases de Python que mediante el uso de *SQLAlchemy* podrán ser persistidos, consultados y actualizados en la Base de datos (BD).

- User: usuario del sistema
- Role: roles del sistema, en este caso Alumno y Administrador
- Team: alumnos que forman un equipo
- ClassGroup: grupo de alumnos que representan una clase en particular
- Badge: Insignia obtenida al completarse objetivos
- Request: envío de solución de un usuario a una Tarea
- Assignment: Tarea a resolver, puede estar asignada a uno o más grupos
- LeaderBoard: marcador de posiciones para una Tarea y Grupo particular
- Attachment: fichero adjunto añadido a una Tarea
- Task: tarea a procesar, utilizada por la Cola interna de *Pizarra*

Los modelos expuestos previamente se pueden encontrar en el fichero **models.py** siguiendo el estándar en *Flask* y se expresan como clases en Python que extienden de la clase *db.Model* del paquete de *SQLAlchemy*, esto permite la serialización del objeto y su persistencia en la Base de datos (BD).

A continuación como ejemplo la representación de un equipo:

```

...
class Team(db.Model):
    """
    Represents a Team of students in the database
    A Student can be a part of only one team
    A Team can have many students
    """
    __tablename__ = 'team'
    id = Column(Integer, primary_key=True)
    name = Column(String, unique=True)

```

```

key = Column(String)
members = relationship('User', back_populates='team')
...

```

models.py

En el código expuesto se puede identificar los siguientes elementos

- la clase *Team* extendiendo la clase *db.Model*
- una descripción de la clase y su propósito
- el nombre de la tabla para persistir el objeto una vez serializado
- las diferentes columnas de la tabla con sus atributos (si es clave primaria, clave única y el tipo de datos)
- la relación entre *Team* y *User*. Un equipo tiene miembros y un usuario es parte de un equipo.

Una vez definidos todos los objetos con sus relaciones podemos generar un diagrama UML de la BD, esto facilita la comprensión y ofrece una mejor visión general.

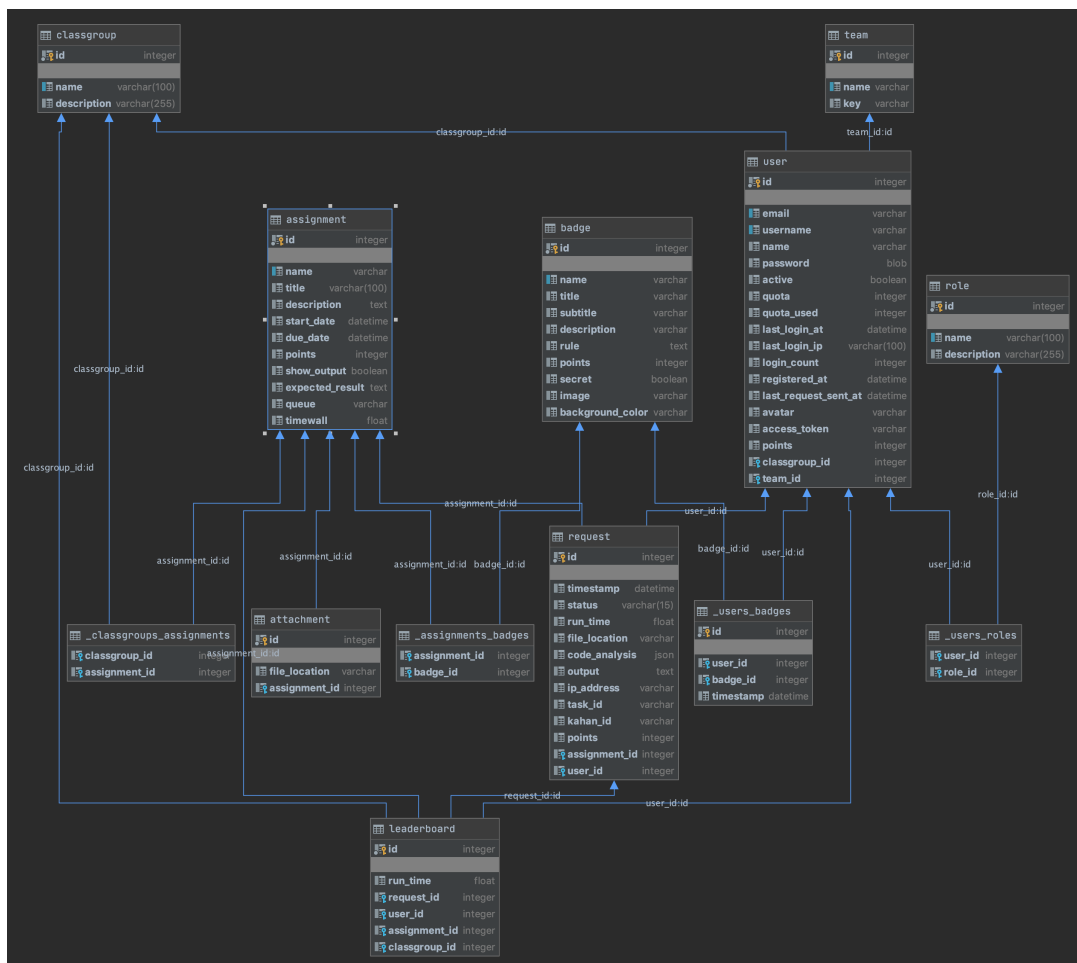


Figura 4.2: Diagrama UML de la Base de Datos

4.4 Envío de soluciones de Tareas

Cuando un Alumno realiza un Envío, esto es, una petición web que incluye el código fuente a compilar y ejecutar como propuesta de solución a una Tarea, la aplicación debe procesar la petición y el Envío pasa por una serie de verificaciones que se describen a continuación, algunas de estas no se realizan para un usuario con rol de administrador del sistema.

1. se autentifica al usuario
2. se busca la Tarea en la BD
3. la Tarea debe estar abierta a recibir Envíos
4. el Alumno pertenece a un Grupo que la tenga asignada
5. el Alumno no ha enviado peticiones muy rápido
6. el Alumno tiene Quota (tiempo disponible de ejecución) para procesar el Envío
7. el Envío contiene un fichero a procesar
8. el fichero a procesar con el código fuente no contiene código malicioso

Si alguno de los pasos anteriores no pasa uno de los puntos de verificación se devuelve un mensaje de error con el código HTTP correspondiente. En el caso de que todas las chequeos sean satisfactorios se crea un objeto de tipo *Request* en la BD y otro de tipo *Task* para la Cola interna en Redis.

El Envío de una Tarea se realiza desde un terminal de comandos u otro cliente que permita enviar peticiones web como *Postman* y está construida para devolver respuestas en formato JSON ya que es un estándar utilizado en muchas aplicaciones web. Un Envío contiene como parámetros el usuario, token de acceso, fichero a procesar con el código fuente y la URL de la Tarea, al procesarse la petición se recibe la respuesta.

A continuación varios ejemplos; Una autenticación fallida, un Envío satisfactorio y un Envío no procesado porque el usuario debe esperar un tiempo.

```
$ curl -k -X POST -F 'file=@primos.c' http://foo:bar@127.0.0.1:5000/assignments/primos
{
  "code": 401,
  "message": "Authentication failed"
}
$ curl -k -X POST -F 'file=@primos.c' http://foo:gB8HSgWANT7x70ipnFIkFQkC1sVLp0@127.0.0.1:5000/assignments/primos
{
  "code": 201,
  "message": "Request created, please navigate to http://127.0.0.1:5000/requests/13 to check the results"
}
$ curl -k -X POST -F 'file=@primos.c' http://foo:gB8HSgWANT7x70ipnFIkFQkC1sVLp0@127.0.0.1:5000/assignments/primos
{
```

```
"code": 403,  
"message": "You are sending Requests too fast. Time between  
Requests is 60 seconds"  
}
```

El diagrama de flujo de una petición web es el siguiente:

4.5 Sistema de colas

Como se vio previamente, *Pizarra* tiene un sistema de Colas interno. Esto se debe a varios motivos de peso, en primer lugar se quiere hacer una separación de roles entre la aplicación web con la que los usuarios interactúan y la ejecución de código fuente. *Pizarra* debe estar preparado para encolar tareas en el cluster kahan del DSIC, pero otra de las posibilidades es la disponibilidad de poder hacerlo de forma local para cualquier otra persona que quiera hacer uso del aplicativo. *Tablón*, el software del cual nos hemos inspirado también ofrece ejecución local, pero con ciertas limitaciones, siendo una de ellas la obligación de reiniciar la aplicación cada 60 minutos, como podemos ver en su apartado de preguntas frecuentes (FAQ)⁵

“El servidor tablón está configurado para reiniciarse cada cierto tiempo. De esta forma se eliminan trabajos erróneos y se eliminan las peticiones http no completadas. El tiempo que resta hasta el próximo reinicio se muestra en la página principal.”

Además, una ejecución local desde el mismo aplicativo obliga a brindarle de recursos necesarios ya sean procesadores y memoria; Esto no es muy eficiente ya que las aplicaciones en Python, y especialmente en *Flask* consumen muy pocos recursos. Por eso mismo, nuestra aplicación se puede ejecutar en dos modos diferentes:

- Modo *Pizarra*: aplicación web con la que interactúan los usuarios
- Modo *Worker*: un proceso que se encarga de revisar en las colas disponibles si hay *Tasks* a procesar

Otro beneficio que brinda esta separación de roles es que se pueden ejecutar un número indefinido de *Workers*, donde cada uno puede tener diferentes colas configuradas y unos recursos disponibles exclusivos para cada uno de ellos. La definición de *Worker* y *Cola* que utilizamos están definidos en la librería RQ (Redis Queue) que permite procesar “trabajos” de forma asíncrona usando como fuente de datos un servicio de *Redis*

Este proceso se puede ver resumido en la siguiente figura.

4.6 Arquitectura de la solución

Ya hemos visto que *Pizarra* no es solo un “sitio web” y está compuesto por varios elementos necesarios para su funcionamiento. Estos son:

- la aplicación web con la que interactúan los usuarios
- la base de datos donde se guarda la información
- el servicio de *Redis* para las Colas
-

⁵Tablón FAQ: <http://frontendv.infor.uva.es/faq>

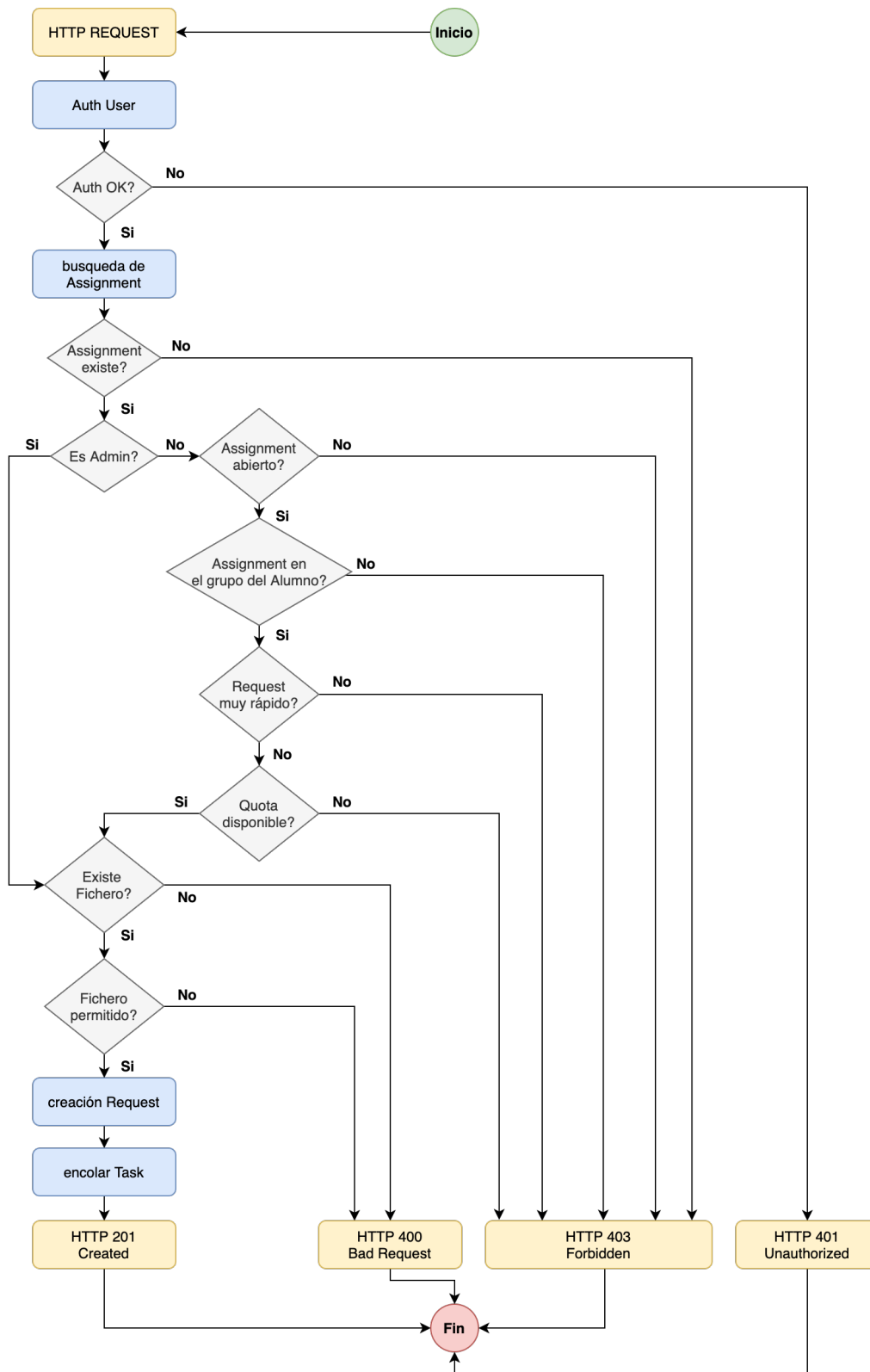


Figura 4.3: Diagrama de flujo de una petición web para un Envío

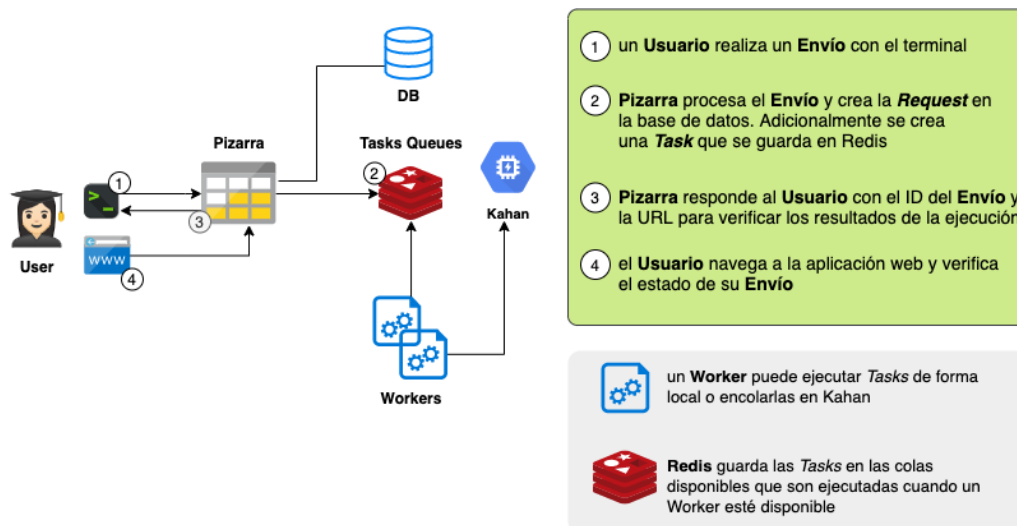


Figura 4.4: Proceso que sigue un Envío en la Cola interna

CAPÍTULO 5

Desarrollo de la solución

?

5.1 Entregables

Sprints ¿?

CAPÍTULO 6

Implantación

Pizarra es una herramienta que necesita de otros aplicativos para su funcionamiento. Por un lado una *BD* para la persistencia de datos, Redis para el sistema de colas interno de envío de tareas, un NFS para compartir recursos y Nginx para actuar como servidor web y redirigir las peticiones. En el pasado la implantación hubiera requerido la intervención del equipo de Administración de Sistemas que se encargue de instalar el *SO* de cada máquina, crear usuarios, asignar permisos, abrir puertos y otro sinfín de tareas.

Para evitar todos estos pasos y acelerar la implantación el proyecto incluye la posibilidad de que todos los elementos de la arquitectura se desplieguen en Contenedores. Estos Contenedores están creados con la tecnología Docker y son paquetes que incluyen todo lo necesario, sean librerías y herramientas del sistema para que el software se ejecute.

Docker funciona de forma similar a una maquina virtual, con algunas ventajas como la asignación de recursos de forma dinámica, espacio en disco reducido ya que podemos evitar la instalación de un *SO* en los contenedores y la portabilidad.

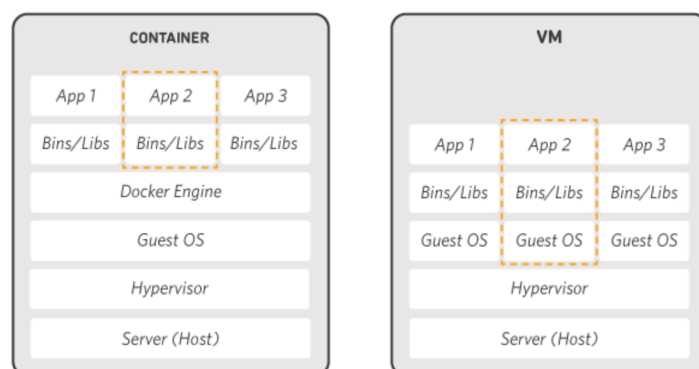


Figura 6.1: Diferencias entre contenedor y máquina virtual

6.1 Docker y Kubernetes

Para nuestro entorno de desarrollo local utilizamos contenedores públicos como el de Postgres que con un fichero de configuración nos permite tener el servicio ejecutándose en cuestión de minutos.

A continuación, un extracto del fichero **docker-compose.yml** del repositorio¹ con una breve explicación de los parámetros.

¹Docker yaml: <https://github.com/nimar3/pizarra/blob/master/docker-compose.yml>

```
1 version: "3.1"
2 services:
3   postgres:
4     image: postgres: 9.6-alpine
5     container_name: postgres
6     volumes:
7       - /data/pizarra/postgres:/var/lib/postgresql/data
8     ports:
9       - 5432: 5432
10    environment:
11      - POSTGRES_USER=pizarra
12      - POSTGRES_PASSWORD=pizarra
13    restart: unless-stopped
```

- **version:** versión de la API de Docker que utilizaremos.
- **services:** listado de servicios que desplegaremos.
- **image:** contenedor que utilizaremos, en este caso el oficial de postgres² sobre Linux Alpine.
- **container_name:** nombre con el que comenzará el contenedor que se ejecute.
- **volumes:** mapeo de directorios de nuestro entorno de desarrollo local al contenedor, esto se hace para evitar perder la información de la *BD* al ser los contenedores stateless.
- **ports:** mapeo de puertos del *SO* a los contenedores, esto permite acceder al servicio de Postgres por un puerto local.
- **environment:** variables de entorno del *SO*, en nuestro caso definimos un usuario y contraseña para acceder a la *BD*
- **restart:** con el parámetro `unless-stopped` nuestro contenedor se reiniciará de forma automática en caso de errores a menos que se envíe un mensaje de finalización de ejecución.

Con los parámetros previos y siguiendo la documentación de variables de entorno disponibles, al iniciarse el contenedor se crearán los schemas y directorios necesarios en caso de que no existan y un usuario con privilegios con el nombre y contraseña proporcionados.

Para un entorno de desarrollo local y/o de pruebas Docker puede ser una herramienta que cumpla con nuestras necesidades, pero para un entorno de producción es necesario contar con un servicio que realice la implantación de forma automatizada, que sea eficiente en el manejo de recursos y resiliente en caso de problemas. Para ésta tarea utilizaremos Kubernetes, un *orquestador* de contenedores que permite auto escalado de aplicaciones, automatización de despliegues y configuración de forma declarativa.

Debido a una limitación de las máquinas donde está alojado kahan no es posible la instalación del servicio de Kubernetes y se ha optado por un despliegue de Pizarra en la nube.

²Postgres Official Docker Image: https://hub.docker.com/_/postgres

6.2 Despliegue en la Nube con Kubernetes

Existen varios proveedores de *K8s* en Internet pudiendo consultar el listado de todos los partners en el sitio oficial³. Siendo los más importantes Google Cloud Engine, Amazon AWS y Microsoft Azure se ha escogido por desplegar la aplicación en *GCE* ya que dispone de buena documentación y además \$300 de crédito promocional que són más que suficientes para tener el aplicativo corriendo durante meses.

6.2.1. Pasos de un despliegue en GCE

Debemos crearnos una cuenta en Google o utilizar una existente que nunca ha activado *GCE* de forma previa ya que el crédito promocional de bienvenida tiene una caducidad de 12 meses desde la activación.

Con nuestra cuenta accedemos a la Consola Web de GCE y creamos nuestro proyecto, en este caso lo llamaremos Pizarra. Nuestro aplicativo utiliza Kubernetes Engine y el Container Registry que son visibles desde el menú lateral, entraremos a cada uno de ellos para habilitar su API.

La imagen muestra dos capturas de pantalla de la interfaz de Google Cloud Platform. La parte superior muestra el 'Dashboard' con un mensaje que indica 'To view this page, select a project.' y un botón 'CREATE PROJECT' resaltado con un recuadro rojo. La parte inferior muestra la pantalla 'New Project' con el 'Project name' establecido como 'pizarra', el 'Project ID' generado como 'pizarra-279707', y la 'Location' configurada como 'No organization'. Los botones 'CREATE' y 'CANCEL' están visibles al final.

Figura 6.2: Creación de un nuevo proyecto en Google Cloud Engine.

Una vez creado el proyecto y habilitadas las API's hay que descargar⁴ e inicializar⁵ Google Cloud SDK para hacer uso de la consola de forma local.

Todos los pasos descritos a continuación se ejecutan de forma automática con un script⁶ incluido en el repositorio de código. Lo único que se necesita de forma previa es la configuración de algunos parámetros de despliegue como variables de entorno del SO donde se ejecuta.

³Kubernetes - Partners: <https://kubernetes.io/es/partners>

⁴Cómo instalar el SDK de Google Cloud: <https://cloud.google.com/sdk/install?hl=es-419>

⁵Cómo inicializar el SDK de Cloud: <https://cloud.google.com/sdk/docs/initializing?hl=es-419>

⁶Script Despliegue GCE: <https://github.com/nimar3/pizarra/blob/master/gce/commands.sh>

Las variables de entorno necesarias son el ID del proyecto en GCE y las credenciales de la BD que serán guardados como un secreto en Kubernetes. Podemos exportarlas antes de lanzar nuestro script ejecutando los siguientes comandos en nuestra consola.

```
1 $ export PROJECT_ID=pizarra-id
2 $ export DB_USERNAME=pizarra
3 $ export DB_PASSWORD=pizarra
```

Éstas son las líneas que ejecuta el script, donde todos los ficheros que forman parte de cada comando se encuentran en el repositorio⁷. Al ser la sintaxis similar a la previamente explicada y evitando que éste documento se extienda de sobremanera al mencionar cada ítem, invitamos al lector a revisar la documentación oficial de Kubernetes⁸.

1. Seteo de proyecto y zona geográfica donde desplegaremos el aplicativo. Por cercanía geográfica se ha escogido el centro de datos de Países Bajos.

```
1 $ gcloud config set project ${PROJECT_ID}
2 $ gcloud config set compute/zone europe-west4
```

2. Credenciales para la BD para ser obtenidas a posterior por los contenedores de Kubernetes

```
1 $ kubectl create secret generic pizarra-credentials \
2 --from-literal db_username=${DB_USERNAME} \
3 --from-literal db_password=${DB_PASSWORD}
```

3. Subida de Contenedores de Nginx y Pizarra al Registro de GCE. Los otros Contenedores utilizarán imágenes públicas.

```
1 $ docker push eu.gcr.io/${PROJECT_ID}/pizarra
2 $ docker push eu.gcr.io/${PROJECT_ID}/nginx
```

4. Creación de cluster en Kubernetes con 2 nodos. Una vez creado se descargan las credenciales para trabajar con él.

```
1 $ gcloud container clusters create pizarra --num-nodes=2
2 $ gcloud container clusters get-credentials pizarra
```

5. Creación de Almacenamiento de 10 GiB para la persistencia de datos que será utilizado por el Sistema de archivos de red. Replicado en varias zonas ya que el almacenamiento no es parte de Kubernetes.

```
1 $ kubectl apply -f storage.yaml
```

6. Creación del servicio de NFS

```
1 $ kubectl apply -f nfs.yaml
```

7. Creación del servicio de Redis para el sistema de colas local

```
1 $ kubectl apply -f redis.yaml
```

8. Creación del servicio de Base de datos

```
1 $ kubectl apply -f postgres.yaml
```

⁷Ficheros de despliegue en GCE: <https://github.com/nimar3/pizarra/tree/master/gce>

⁸Documentación Kubernetes: <https://kubernetes.io/docs>

9. Despliegue de Pizarra y 1 Worker que se encargará de encolar las las Tareas en kahan

```
1 $ kubectl apply -f pizarra.yaml
```

10. Despliegue del servicio de NGinx con una IP pública para el acceso externo de usuarios

```
1 $ kubectl apply -f nginx.yaml
```

Una vez completados todos los pasos se puede obtener la IP pública de Pizarra ejecutando el siguiente comando.

```
1 $ kubectl get services
```

También se puede obtener en la web de GCE sobre la pestaña Services e Ingress.

IMAGEN

Hemos completado el despliegue y ya podemos navegar a Pizarra para comenzar a interactuar con el aplicativo.

IMAGEN

6.2.2. Costos asociados

GCE cuenta con una herramienta⁹ para estimar el gasto mensual en el que podemos incurrir con la utilización de los recursos contratados. Se ha realiado una estimación con los siguientes parámetros:

- Kubernetes
 - Número de nodos: 2
 - Tipo de Instancia: n1-standard-n1 (vCPUs: 1, RAM: 3.75 GB)
 - Centro de Datos: Países Bajos
 - Uso estipulado: 24 horas al día los 7 días a la semana.
- Persistencia
 - Tamaño disco: 10 GiB
 - Centro de Datos: Países Bajos

En resumen, el número de nodos es 2 para tener servicio en caso de que haya una caída de uno de los nodos del cluster, el centro de datos se ha escogido Países Bajos por cercanía geográfica y una instancia con los recursos mínimos necesarios para los contenedores. El disco es compartido entre la *BD* y Pizarra con suficiente margen para evitar problemas de espacio.

En total llegamos a un costo estimado de €49.00 mensuales. Como dato adicional, se puede variar de Centro de Datos y otros parámetros como el uso estipulado para reducir los costos aún más. Podemos guardar la estimación o enviárnosla por correo electrónico.

⁹Calculadora de precios de Google: <https://cloud.google.com/products/calculator/>

Kubernetes Engine

Number of nodes *
2

Number of Zonal clusters - excluding Anthos GKE clusters ^{1,2}

Number of Regional clusters - excluding Anthos GKE clusters ¹

What are these nodes for?

Instance type
n1-standard-1 (vCPUs: 1, RAM: 3.75 GB)

☐ Add GPUs.

Local SSD
0

Datacenter location
Netherlands (europe-west4)

Committed usage
None

Average hours per day each node is running *
24 hours per day

Average days per week each node is running *
7


¹ Anthos GKE clusters are exempted from GKE Cluster Management Fee.
² One Zonal cluster is free per billing account.

Persistent Disk

Location
Netherlands (europe-west4)

Persistent disk storage
10 GiB

Figura 6.3: Configuración de la calculadora de precios para el despliegue en Kubernetes de Pizarra.

 **Google Cloud**

Nicolas Martini,

Your Estimated Bill *

Estimated Monthly Cost: EUR 49.00



 2 x	n1-standard-1	1460 total hours per month	EUR 48.60
 Persistent disk	Storage	10 GiB	EUR 0.40
Total Estimated Monthly Cost			EUR 49.00

Figura 6.4: Ejemplo de correo de GCE con una estimación de los costos asociados por mes para la implantación.

CAPÍTULO 7

Mantenimiento

El desarrollo de software es un proceso evolutivo y requiere de un mantenimiento continuo para asegurar el correcto funcionamiento, como así también la posibilidad de añadir mejoras y nuevas funcionalidades. Para poder asegurar este proceso tienen que cumplirse ciertas pautas.

Acceso al código fuente, en este caso disponible en un repositorio de código abierto en Internet, una guía de como configurar un entorno de desarrollo local y el proceso a seguir para mantener un estándar de calidad en el código fuente. Los siguientes apartados abordan todas estas pautas y así ayudar con este proceso.

7.1 Entorno local de desarrollo

Para configurar un entorno de desarrollo local se debe cumplir los siguientes requisitos.

- Tener instalado Python3¹ y Docker² en nuestra máquina
- Clonado el repositorio de Pizarra

1. Creación de directorio

```
1 $ mkdir pizarra
2 $ cd pizarra/
```

2. Inicialización de un repositorio nuevo y la dirección remota

```
1 $ git init .
2 Initialized empty Git repository in /Users/nmartini/pizarra/.git/
3 $ git remote add upstream git@github.com:nimar3/pizarra.git
```

3. Descarga del código fuente

```
1 $ git fetch upstream
2 remote: Enumerating objects: 589, done.
3 remote: Counting objects: 100% (589/589), done.
4 remote: Compressing objects: 100% (364/364), done.
5 remote: Total 6891 (delta 396), reused 393 (delta 221), pack-
   reused 6302
6 Receiving objects: 100% (6891/6891), 19.14 MiB | 675.00 KiB/s,
   done.
7 Resolving deltas: 100% (1748/1748), done.
```

¹Descarga oficial de Python: <https://www.python.org/downloads/>

²Descarga oficial de Docker: <https://docs.docker.com/get-docker/>

4. checkout a la rama de desarrollo

```

1 $ git checkout master
2 Checking out files: 100% (4858/4858), done.
3 Branch 'master' set up to track remote branch 'master' from '
4 upstream'.
   Already on 'master'

```

- Creación de un entorno virtual e instalación de todos los Paquetes de Python necesarios

```

1 $ python3 -m venv env
2 $ source env/bin/activate
3 (env) $ pip3 install -r requirements_dev.txt

```

Si hemos realizado cambios en el código fuente se debe ejecutar el siguiente comando para generar una nueva versión de nuestro Contenedor de Docker de Pizarra

```

1 $ docker build -t pizarra .

```

Por último, para ejecutar Pizarra con todos los componentes introducimos el siguiente comando en la raíz del repositorio. Todos los componentes deben mostrar que se han creado correctamente con el mensaje *done*

```

1 (env) $ docker-compose up
2 Creating network "pizarra_default" with the default driver
3 Creating redis ... done
4 Creating postgres ... done
5 Creating pizarra ... done
6 Creating worker ... done
7 Creating nginx ... done

```

Se puede acceder a la aplicación web con mediante la URL <https://127.0.0.1/> y se redirigirá automáticamente al inicio de sesión.

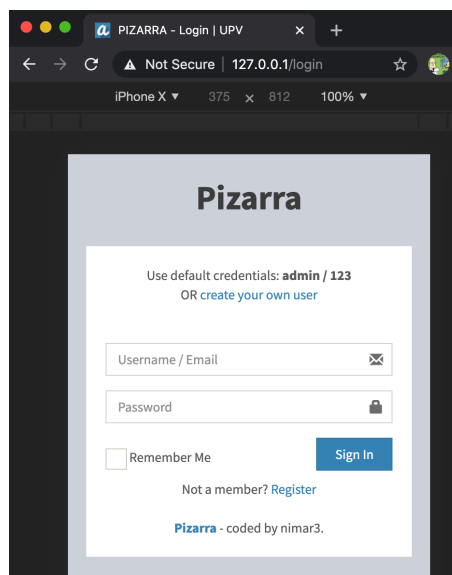


Figura 7.1: Inicio de sesión en Pizarra.

En algunos casos sólo es necesario la ejecución del aplicativo web con una base de datos local en SQLite. Para estos casos se puede utilizar el siguiente comando y acceder mediante la URL <http://127.0.0.1:5000/> (el puerto por defecto de *Flask* es el 5000).


```
1 (env) $ python3 run.py
2 * Serving Flask app "app" (lazy loading)
3 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

7.2 Depuración de errores

Como todo software, Pizarra no estará exento de errores o bugs que no se han identificado en la etapa de desarrollo y como muchos otros lenguajes Python ofrece la ejecución de código en modo Debug, lo que nos permite poner puntos de ruptura y hacer una ejecución *paso a paso* del código para identificar el problema.

En el entorno de desarrollo local, el modo Debug siempre está habilitado. Si queremos habilitarlo cuando ejecutamos el aplicativo en modo *Production* debemos especificarlo explícitamente en las variables de entorno del SO, en este caso y para más información revisar el Apéndice A: Configuración del Sistema.

7.3 Calidad de código

Para una mejor lectura del código fuente se ha utilizado el Paquete *flake8*³ que nos obliga a seguir las guías de estilo oficiales de Python.

En el directorio raíz del repositorio de código ejecutamos el siguiente comando y podemos ver un listado de errores a corregir.

```
1 $ flake8 app/
2 app/__init__.py:8:1: F401 'logging.DEBUG' imported but unused
3 app/__init__.py:71:5: E722 do not use bare 'except'
4 app/home/routes.py:59:101: E501 line too long (106 > 100 characters)
5 app/home/routes.py:210:5: E722 do not use bare 'except'
6 app/home/routes.py:216:101: E501 line too long (108 > 100 characters)
7 app/admin/forms.py:32:28: N805 first argument of a method should be named 'self'
```

La configuración de *flake8* se encuentra en el fichero **setup.tfg** en la raíz del repositorio.

```
1 [flake8]
2 max-line-length=100
3 ignore=E402,E266
4 exclude=./migrations
```

Otro punto importante es la cobertura del código fuente. Aunque el repositorio incluye las librerías de *coveralls* y *pytest* por cuestiones de tiempo no se han escrito tests unitarios y se ha dejado como una mejora futura.

³Sitio Web de Flake8: <https://flake8.pycqa.org/en/latest/>

CAPÍTULO 8

Extensibilidad

Uno de los puntos más importantes es permitir la colaboración y que Pizarra pueda utilizar otros sistemas de tareas que no sean una Cola local o la de un cluster como kahan (TORQUE¹) así como también que pueda ser localizado en diferentes idiomas.

8.1 Otros sistemas de Gestión de Tareas

La ejecución de una tarea es diferente en cada Cola y para esto se ha implementado un autómatas que va cambiando de estados en cada ejecución hasta finalizarla. Esto permite que cada nuevo tipo de Cola tenga su propia definición, sea flexible y compatible con el funcionamiento del aplicativo.

La definición de estas Colas se encuentra en **models_tasks.py**² siendo *Task* el objeto a extender para las nuevas implementaciones.

```
1 class Task:
2
3     def __init__(self, user_request):
4         self.user_request = user_request
5         self.output = ""
6         self.binary_file_location = ""
7         self.return_code = 0
8         self.run_time = 0.0
9         self.points_earned = 0
10        # automata
11        self.task_process = {}
```

El autómatas cuenta con las siguientes transiciones

```
1 class StepResult(enum.Enum):
2     START = 0
3     OK = 1
4     NOK = 2
5     WAIT = 3
6     END = 4
```

Para el sistema de Colas de kahan tenemos el siguiente extracto. Donde creamos la nueva clase *KahanTask* que extiende de *LocalTask* y hereda todos sus atributos y métodos.

```
1 class KahanTask(LocalTask):
```

¹Sitio web de TORQUE: <https://adaptivecomputing.com/cherry-services/torque-resource-manager/>

²Definición de Colas: https://github.com/nimar3/pizarra/blob/master/app/base/models_tasks.py

```

2
3 def __init__(self, user_request):
4     super().__init__(user_request)
5     self.task_process = {
6         RequestStatus.CREATED: {
7             'f': self.start,
8             'steps': {
9                 StepResult.OK: RequestStatus.VERIFYING,
10            }
11        },
12        RequestStatus.VERIFYING: {
13            'f': self.verify,
14            'steps': {
15                StepResult.OK: RequestStatus.COMPILING,
16                StepResult.NOK: RequestStatus.ERROR
17            }
18        },
19        RequestStatus.COMPILING: {
20            'f': self.compile,
21            'steps': {
22                StepResult.OK: RequestStatus.DEPLOYING,
23                StepResult.NOK: RequestStatus.ERROR
24            }
25        },
26        ...

```

En la definición del `task_process` tenemos un diccionario con los diferentes estados que puede encontrarse el autómata, la función a ejecutar `f` y para cada ejecución el diccionario `steps` con las transiciones.

8.2 Localización

Pizarra utiliza el Paquete *Babel*³ para ofrecer los diferentes idiomas disponibles a los usuarios. A continuación se muestra el *paso a paso* para añadir un nuevo Idioma.

La configuración de *Babel* se encuentra en el fichero **babel.cfg**⁴ en la que se define los ficheros a escanear y extraer texto a traducir.

```

1 [python: app/**/*.py]
2 [jinja2: app/**/*.templates/**/*.html]
3 extensions=jinja2.ext.autoescape, jinja2.ext.with_

```

Con el siguiente comando se extrae el texto. En la consola se visualiza como se van escaneando todos los ficheros y el resultado lo tendremos disponible en el fichero **messages.pot**

```

1 (venv) $ pybabel extract -F babel.cfg -k _l -o messages.pot .
2 ...
3 extracting messages from app/home/templates/macros/summaries.html (extensions="
4     jinja2.ext.autoescape, jinja2.ext.with_)
5 extracting messages from app/home/templates/macros/teams.html (extensions="
6     jinja2.ext.autoescape, jinja2.ext.with_)
7 writing PO template file to messages.pot

```

Una vez extraído el texto hay que generar el fichero de localización para el idioma que vayamos a traducir. Las localizaciones se encuentran en el directorio `/app/translations`.

```

1 (env) $ pybabel init -i messages.pot -d app/translations -l es

```

³Paquete *Babel*: <http://babel.pocoo.org/en/latest/>

⁴Configuración de *Babel*: <https://github.com/nimar3/pizarra/blob/master/babel.cfg>

```
2 creating catalog app/translations/es/LC_MESSAGES/messages.po based on messages.pot
```

El siguiente paso es comenzar a añadir las traducciones en las líneas identificadas con *msgstr*

```
1 $ cat app/translations/es/LC_MESSAGES/messages.po
2 ...
3 #: app/account/routes.py:34
4 msgid "New Access Token has been generated!"
5 msgstr "Nuevo Token de acceso ha sido generado!"
6
7 #: app/account/routes.py:54
8 msgid "You joined Team {}"
9 msgstr "Te has unido al equipo {}"
10
11 #: app/account/routes.py:56
12 msgid "You are unable to join team {} because is full"
13 msgstr ""
14 ...
```

Se deben compilar las traducciones.

```
1 (env) $ pybabel compile -d app/translations
2 compiling catalog app/translations/es/LC_MESSAGES/messages.po to app/
   translations/es/LC_MESSAGES/messages.mo
```

Y se añade a la configuración de Pizarra el nuevo idioma disponible.

```
...
SUPPORTED_LANGUAGES = {'es': 'Spanish', 'en': 'English'}
...
```

config.py

En el menú se visualizará el nuevo idioma disponible.

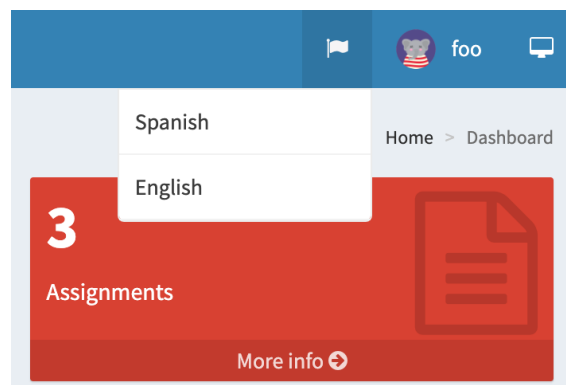


Figura 8.1: Selector de Idiomas en Pizarra.

CAPÍTULO 9

Conclusiones

...

9.1 Relación del trabajo desarrollado con los estudios cursados

?

CAPÍTULO 10

Trabajos futuros

?

Bibliografía

- [1] Jennifer S. Light. When computers were women. *Technology and Culture*, 40:3:455–483, juliol, 1999.
- [2] Georges Ifrah. *Historia universal de las cifras*. Espasa Calpe, S.A., Madrid, sisena edició, 2008.
- [3] Comunicat de premsa del Departament de la Guerra, emés el 16 de febrer de 1946. Consultat a <http://americanhistory.si.edu/comphist/pr1.pdf>.

APÉNDICE A

Configuración del sistema

Para un correcto funcionamiento de Pizarra el sistema debe inicializarse siguiendo los pasos descritos a continuación. Aunque se dispone de un script que realiza el despliegue de forma automatizada hay algunos pasos que requieren de intervención manual.

A.1 Inicialización

Antes de comenzar con la inicialización del sistema, debemos exportar las variables de entorno del SO que utiliza el script. Abrimos una consola y ejecutamos los siguientes comandos, donde *pizarra-id* es el ID de proyecto en Google Cloud Engine (GCE).

```
1 $ export PROJECT_ID=pizarra-id
2 $ export DB_USERNAME=pizarra
3 $ export DB_PASSWORD=pizarra
```

Si quisiéramos cambiar algún parámetro por defecto de Pizarra, debemos modificar el fichero **pizarra.yaml**¹ añadiendo los nuevos parámetros. En el siguiente extracto se han cambiado los puntos de penalización al obtener un *KO* y *TIMEWALL* en el envío de una Tarea y la carga de datos de ejemplo.

```
1 ...
2 spec:
3   containers:
4     - image: eu.gcr.io/pizarra-279100/pizarra
5       name: pizarra
6       env:
7         - name: TIMEWALL_PENALTY
8           value: "-20"
9         - name: "KO_PENALTY"
10          value: "-30"
11         - name: "IMPORT_SAMPLE_DATA"
12          value: "True"
13 ...
```

A continuación se ejecuta el script.

```
1 $ sh commands.sh
```

¹Fichero YAML de Pizarra: <https://github.com/nimar3/pizarra/blob/master/gce/pizarra.yaml>

Al terminar la ejecución deberíamos tener Pizarra con todos los componentes desplegados. Podemos obtener el estado de los contenedores y servicios con los siguientes comandos.

```
1 $ kubectl get pods
2 $ kubectl get services
```

A.2 Parámetros

Al iniciar un Contenedor utilizamos variables de entorno del SO para configurar su funcionamiento. El Contenedor de Pizarra soporta estas variables para dar diferentes posibilidades en cómo queremos que se comporte el aplicativo.

Los mencionados a continuación con su descripción y opciones son los más relevantes en un despliegue. También se listan otros parámetros adicionales como referencia. Todos estos pueden ser consultados en el fichero de configuración².

■ APP_MODE

- Descripción: ejecución del aplicativo web o un worker
- Opciones
 - Pizarra: aplicativo web
 - Worker: worker
- Valor por defecto: *Pizarra*

■ CONFIG_MODE

- Descripción: modo en el cual queremos que se ejecute Pizarra
- Opciones
 - Debug: modo Debug habilitado y Base de datos (BD) SQLite
 - Production: modo Debug deshabilitado y Base de datos (BD) Postgres
- Valor por defecto: *Debug*

■ REMOTE_HOST

- Descripción: hostname o IP del sistema externo a conectarnos para las colas de kahan
- Valor por defecto: *kahan.dsic.upv.es*

■ REMOTE_USER

- Descripción: usuario con el que nos conectaremos por SSH y SCP a kahan
- Valor por defecto: *pizarra*

■ REMOTE_PATH

- Descripción: directorio remoto donde se copiarán los ficheros de cada ejecución en kahan
- Valor por defecto: */pizarra*

²Fichero de Configuración: <https://github.com/nimar3/pizarra/blob/master/config.py>

■ SSH_FILE_PATH

- Descripción: ubicación de la clave privada para conectarnos por SSH y SCP a kahan
- Valor por defecto: *app/data/keys/id_rsa*

■ LOG_LEVEL

- Descripción: establece el nivel de mensajes que deben mostrarse al ejecutarse el aplicativo
- Opciones
 - CRITICAL
 - ERROR
 - WARNING
 - INFO
 - DEBUG
- Valor por defecto: *INFO*

■ SECRET_KEY

- Descripción: clave secreta con la que se encriptarán las contraseñas, no se puede cambiar una vez desplegada la aplicación y por seguridad nunca debe utilizarse el valor por defecto
- Valor por defecto: *pizarra-app*

■ IMPORT_SAMPLE_DATA

- Descripción: borrado de información de la Base de datos e importación de data de ejemplo.
- Valor por defecto: *False*

■ TIME_BETWEEN_REQUESTS

- Descripción: tiempo mínimo en segundos que debe esperar un Alumno entre cada envío de Tareas
- Valor por defecto: *60*

■ TEAM_MAX_SIZE

- Descripción: tamaño máximo de Alumnos que puede tener un Equipo
- Valor por defecto: *3*

■ REGISTRATION_ENABLED

- Descripción: habilita el registro de usuarios en el aplicativo
- Valor por defecto: *False*

■ TIMEWALL_PENALTY

- Descripción: puntos de penalización al obtener un resultado de *TIMEWALL* en un envío de Tarea

- Valor por defecto: *-10*
- **KO_PENALTY**
 - Descripción: puntos de penalización al obtener un resultado de *KO* en un envío de Tarea
 - Valor por defecto: *-15*
- **DEBUG**
 - Descripción: inicia el aplicativo en modo Debug, útil para identificar errores.
 - Valor por defecto: *False*

Parámetros adicionales

- SQLALCHEMY_DATABASE_URI
- SQLALCHEMY_TRACK_MODIFICATIONS
- JSONIFY_PRETTYPRINT_REGULAR
- SUPPORTED_LANGUAGES
- BABEL_DEFAULT_LOCALE
- BABEL_DEFAULT_TIMEZONE
- UPLOAD_FOLDER
- FILE_ALLOWED_EXTENSIONS
- MAX_CONTENT_LENGTH
- TIMEWALL
- FORBIDDEN_CODE
- RQ_DASHBOARD_REDIS_URL
- QUEUES
- COMPILER

APÉNDICE B

Licencia

Como este es un proyecto de código abierto, se ha escogido la Licencia MIT¹ ya que es una de las más permisivas para fomentar la reutilización y colaboración. Este tipo de licencia permite el libre uso de forma privada o comercial, la modificación y distribución del código sin ningún tipo de garantía donde solo se pide que se mantengan las menciones a la licencia y derechos de autor.

¹Licencia MIT: <https://opensource.org/licenses/MIT>

Acrónimos

- API** Interfaz de programación de aplicaciones. 21, 43, 45
- BD** Base de datos. 11, 12, 15, 16, 22, 40, 41, 43, 45
- CPA** Computación Paralela. 43, 45
- DSIC** Departamento de Sistemas Informáticos y Computación. 43, 45
- ETSINF** Escuela Técnica Superior de Ingeniería Informática. 43, 45
- FIFO** Primero en entrar, primero en salir. 43, 45
- GCE** Google Cloud Engine. 21–23, 39, 43, 45
- GiB** Gibibyte. 22, 43, 45
- K8s** Kubernetes. 43, 45
- LPP** Lenguajes y Entornos de la Programación Paralela. 43, 45
- MVC** Modelo Vista Controlador. 13, 43, 45
- NFS** Sistema de archivos de red. 11, 19, 22, 43, 45
- RAM** Memoria de acceso aleatorio. 43, 45
- RF** Requisito funcional. 43, 45
- RNF** Requisito no funcional. 43, 45
- SCP** Secure Copy. 13, 40, 41, 43, 45
- SO** Sistema operativo. 19, 21, 27, 39, 40, 43, 45
- SSH** Secure Shell. 13, 40, 41, 43, 45
- TFG** Trabajo Final de Grado. 43, 45
- UPV** Universitat Politècnica de València. 43, 45
- vCPUs** Unidades centrales de procesamiento virtuales. 43, 45

Términos

- Alumno** Estudiante que forma parte de un curso académico. 16, 41, 43, 45
- Clase** Plantilla para la creación de objetos de datos según un modelo predeterminado. 43, 45
- Cola** Estructura de datos que sigue una filosofía FIFO. 11, 13, 29, 43, 45
- Contenedor** imagen ejecutable ligera y portátil que contiene el software y todas sus dependencias. 19, 22, 26, 40, 43, 45
- Entorno de producción** lugar donde se ejecuta el aplicativo para los usuarios finales. 43, 45
- Envío** Petición web que incluye el código fuente a compilar y ejecutar como propuesta de solución a una Tarea. 16, 43, 45
- Equipo** Grupo de estudiantes que pertenecen a una misma clase en un grupo académico. 41, 43, 45
- Equipo de Estudiantes** Grupo de uno o más estudiantes formado para resolver Tareas de forma conjunta. 43, 45
- formula** A mathematical expression. 43, 45
- GCD** Greatest Common Divisor. 43, 45
- Grupo** Agrupación de alumnos que pertenecen a un grupo académico. 15, 43, 45
- Grupo de Estudiantes** Clase a la que pertenece el estudiante para el año lectivo. 43, 45
- Insignia** Trofeo o Marcador que se obtiene al realizar cierta acción en la aplicación. 13, 15, 43, 45
- latex** Is a mark up language specially suited for scientific documents. 43, 45
- LCM** Least Common Multiple. 43, 45
- mathematics** Mathematics is what mathematicians do. 43, 45
- Paquete** Carpeta que contiene varios módulos de Python. 11, 12, 15, 26, 27, 30, 43, 45
- Tarea** Problema a resolver asignado a un estudiante o equipo. 11, 13, 15, 16, 23, 39, 41–43, 45