1. Google App Engine.

En este apartado de la memoria voy a explicar lo que es, la configuración y el como usar la plataforma Google App Engine.

1.1. Introduccion.

Google App Engine es una conjunto de apis que proporciona Google para construir tus propias aplicaciones web, que pueden ser alojadas y usadas en su servicio Google App y vendidas en Google Apps Marketplace. Además de alojamiento gratuito Google, ofrecen un dominio, que es: *nombre_de_la_aplicacion.appspot.com* y una base de datos propietaria de Google que se accede transparentemente a través de la api, gestión de usuarios mediante autentificación con cuentas Google del tipo:*usuario@gmail.com*, autentificación por federación o *openID*.

Además de todas esas características Google proporciona apis para Java, Python y Go, este último un lenguaje experimental del propio Google. Para usar dicha API, Google también da un plugin para Eclipse, en caso de que el lenguaje elegido sea Java, que ayuda al despliegue de la aplicación web, autocompletado y gestión de de las aplicaciones creadas.

En el proyecto solo he usado la API de Google App Engine de Java, por lo que todo lo que puedo comentar es de dicha API, la parte de Python y Go no se han estudiado.

En general el uso de Google App Engine para crear aplicaciones web es idéntico a crear una aplicación web con Java 2 Enterprise Edition (Java2EE), se pueden crear servlet que recogen valores **GET** o **POST** y además clases java para hacer operaciones con dichos valores. A su vez para mostrar la infomación se pueden generar archivos **.jsp*, que son archivos html con bloques o líneas de código java que se introducen con estas etiquetas: <%= línea de código Java %> o <% Bloque de código Java %>. A parte de archivos **.java* y **.jsp*, debemos tener una carpeta llamada *war* en la que tiene que ir toda la información de la aplicación web que queremos deplegar. En dicha carpeta hay varias subcarpetas como pueden ser *css* en la que tiene que ir el estilo de la web o *WEB-INF* en la que están todos los archivos de configuración, como pueden ser los permisos que tenemos que tener para poder acceder al uso de un servlet, si la web tiene conexión https, la configuración de la base de datos, etc.

Para este proyecto se han tenido que desarrollar dos aplicaciones web, una que es un servidor de timestamp y otra que es una aplicación para gestión de las firmas digitales que realice cada usuario. A continuación vamos a explicar en profundida la tecnología usada y ambas aplicaciones web.

1.2. Explicación de una aplicación web.

En esta parte voy a explicar en profundidad que es un servlet, los archivos de configuración, los archivos **.jsp* y el resto de archivos necesarios para poder desplegar una aplicación en Google Apps.

1.2.1. ¿Qué es un servlet?

Un servlet es la evolución de los antiguos applet, su uso más comun es generar páginas web dinámente con los parámetros que recibe mediante una petición realizada por el navegador web y datos que están almacenados en el servidor web.

Un servlet es un objeto java que tiene que ser ejecutado en un servidor web o contenedor J2EE, que recibe unos parámetros, realiza una o varias acciones y devuelve un resultado que puede ser desde un código html, un JSP que genera dinámicamente un código html, un JSON o una simple cadena de texto.

Los servlets, junto con JSP, son la solución de Oracle a la generación de contenido dinámico equivalente al lenguaje PHP, ASP de Microsoft, Ruby, etc.

Los servlet forman parte de Java Enterprise Edition (JEE) que a su vez es una amplicación de Java Standard Edition (JSE), para usarlos necesitas un servidor web que pueda interpretar código java, el más famoso es Tomcat que es una adaptación del servidor web Apache, pero como veremos en este proyecto no es el único, ya que el propio Google Apps también funciona a base de servlets y JSP.

Para crear un servlet hay que generar una clase java que implemente la interfaz *javax.servlet.Servlet* o que extienda cualquier clase que herede de una clase o que implemente la anterior interfaz, como puede ser *javax.servlet.http.HttpServlet* que es específico para conexiones HTTP. Una vez generada la clase hay que implementar el método **doGet** para peticiones tipo **GET** o el método **doPost** para peticiones de tipo **POST**. En el siguiente trozo se código se puede ver la implementación mas básica de los métodos **doGet** y **POST**, con las llamadas sus respectivas llamadas a *super*.

```
@Override
protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletExcepti
// TODO Auto-generated method stub
super.doGet(req, resp);
}

@Override
protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletExcept
// TODO Auto-generated method stub
super.doPost(req, resp);
}
```

Una vez implementados los métodos que se necesiten se pueden usar el parámetro *HttpServletRequest req* para recibir los valores que queramos enviar a la aplicación web y podemos usar *HttpServletResponse resp* para enviar lo que queramos desde una redirección a JSP o una página web a una JSON o cadena de texto. Un ejemplo de como se reciben los parámetros sería:

```
String num_sec = req.getParameter("sec");
```

Y si queremos enviar algo por *HttpServletResponse resp* podríamos usar:

```
PrintWriter out = resp.getWriter();
out.print(jsonArray);
out.flush();
```

Como podemos ver el objeto *resp* nos da la posibilidad de conseguir un objeto *java.io.PrintWriter* por el que podemos enviar lo que queramos.

La forma de acceder a un servlet mandandole peticiones **GET** sería la siquiente: https://servertimestamp.appspot.com/search?id=63texto=Prueba