UNIVERSIDAD DE BURGOS

Escuela Politécnica Superior

Gº en Ingeniería Informática

****

**TFG Ingeniería Informática:**

**GII17.0T Trayectorias Semánticas**

Presentado por Hector Cogollos Adrian

en Burgos el 2 julio de 2019

Tutores D. Bruno Baruque Zanón

y D. Santiago Porras Alfonso

# Índice General

[Índice General 1](#_Toc12910439)

[Índice de figuras 2](#_Toc12910440)

[Plan de Proyecto Software 4](#_Toc12910441)

[A.1. Introducción 4](#_Toc12910442)

[A.2. Planificación temporal 4](#_Toc12910443)

[Sprint 1 4](#_Toc12910444)

[Sprint 2 5](#_Toc12910445)

[Sprint 3 5](#_Toc12910446)

[Sprint 4 5](#_Toc12910447)

[Sprint 5 5](#_Toc12910448)

[Sprint 6 5](#_Toc12910449)

[Sprint 7 6](#_Toc12910450)

[Sprint 8 6](#_Toc12910451)

[Sprint 9 6](#_Toc12910452)

[Sprint 10 6](#_Toc12910453)

[Sprint 11 6](#_Toc12910454)

[Sprint 12 7](#_Toc12910455)

[Sprint 13 7](#_Toc12910456)

[Sprint 14 7](#_Toc12910457)

[Sprint 15 7](#_Toc12910458)

[A.3. Estudio de viabilidad 8](#_Toc12910459)

[Viabilidad económica 8](#_Toc12910460)

[Viabilidad legal 9](#_Toc12910461)

[Especificación de Requisitos 11](#_Toc12910462)

[B.1. Introducción 11](#_Toc12910463)

[B.2. Objetivos generales 11](#_Toc12910464)

[B.3. Catálogo de requisitos 11](#_Toc12910465)

[Requisitos funcionales 11](#_Toc12910466)

[Requisitos no funcionales 12](#_Toc12910467)

[Limitaciones Funcionales 12](#_Toc12910468)

[B.4. Especificación de requisitos 13](#_Toc12910469)

[Actores 13](#_Toc12910470)

[Diagramas de casos de uso 13](#_Toc12910471)

[Especificación de los casos de uso 14](#_Toc12910472)

[Especificación de diseño 18](#_Toc12910473)

[C.1. Introducción 18](#_Toc12910474)

[C.2. Diseño de datos 18](#_Toc12910475)

[C.3. Diseño arquitectónico 19](#_Toc12910476)

[Modelo 20](#_Toc12910477)

[Controlador 20](#_Toc12910478)

[Vista 22](#_Toc12910479)

[Base de Datos 23](#_Toc12910480)

[C.4. Diseño procedimental 24](#_Toc12910481)

[SQL 24](#_Toc12910482)

[Predicción 25](#_Toc12910483)

[Vista 26](#_Toc12910484)

[Documentación técnica de programación 28](#_Toc12910485)

[D.1.Introducción 28](#_Toc12910486)

[D.2. Estructura de directorios 28](#_Toc12910487)

[D.3. Manual del programador 28](#_Toc12910488)

[D.4. Compilación, instalación y ejecución del proyecto 28](#_Toc12910489)

[D.5. Pruebas del sistema 28](#_Toc12910490)

[Documentación de usuario 29](#_Toc12910491)

[E.1. Introducción 29](#_Toc12910492)

[E.2. Requisitos de usuarios 29](#_Toc12910493)

[E.3. Instalación 29](#_Toc12910494)

[E.4. Manual del usuario 29](#_Toc12910495)

# Índice de figuras

# Plan de Proyecto Software

## A.1. Introducción

En este apartado se expone la planificación temporal del proyecto y el estudio de viabilidad económico y legal. En el apartado de planificación temporal se explica que técnicas se han utilizado y como se han utilizado. En viabilidad económica veremos los costes de la puesta en marcha del proyecto y en viabilidad legal estudiaremos si las licencias de todos los componentes del proyecto son compatibles legalmente.

## A.2. Planificación temporal

La planificación temporal está basada en **Scrum** una metodología ágil para la gestión de proyectos. Este ha sido adaptado a las circunstancias del proyecto ya que la disponibilidad horaria no ha sido la misma a lo largo del desarrollo y a el hecho de que nos encontramos en un contexto educativo. En este contexto no hay un cliente y tampoco hay reuniones diarias ya que solo hay un desarrollador.

Algunas de las cosas que hay que tener en cuenta antes de pasar a ver los detalles de la planificación son las siguientes:

* El proyecto se divide en sprints o hitos, cada hito tiene una duración similar, aunque adecuándose a las circunstancias.
* Los sprints tienen tareas asociadas con un cálculo de duración en fracciones de tiempo indeterminadas.
* Se persigue que cada sprint tenga una carga de trabajo similar a la del resto de sprints.
* No se ha podido determinar el grado de cumplimiento de los tiempos debido a la falta de horarios.

### Sprint 1

Su finalidad es investigar acerca del proyecto aspectos tales como artículos sobre el tema de las trayectorias semánticas, aplicaciones relacionadas, herramientas para su desarrollo, etc.

Fecha de fin: 30/11/2018

|  |  |
| --- | --- |
| Tarea | Estimación |
| Filtrar trayectorias por tiempo | 5 |
| Extraer datos de las rutas | 3 |
| Estudiar proyectos similares | 3 |
| Estudiar artículos de trayectorias | 13 |
| Investigar herramientas | 8 |
| Añadir memoria al repositorio | 1 |

## Sprint 2

En este sprint se buscaba prepara algunos de los elementos básicos para la aplicación. Fundamentalmente los sistemas de bases de datos y las bases de datos geoespaciales.

Fecha de fin: 17/12/2018

|  |  |
| --- | --- |
| Tarea | Estimación |
| Documentarse sobre la instalación de PostGIS | 3 |
| Instalar PostGIS | 3 |
| Estudiar como cagar datos en PostGIS | 5 |
| Buscar información de como cargar los datos de OSM | 5 |
| Cargar datos de OSM | 3 |

## Sprint 3

Investigar los datos de OSM para ver cuáles son útiles para nuestro proyecto y tratar de construir trayectorias

Fecha de fin: 11/01/2019

|  |  |
| --- | --- |
| Tarea | Estimación |
| Seleccionar datos relevantes de OSM | 8 |
| Anotar Trayectorias | 5 |

Sprint 4

Procesar las rutas para ver que tenemos y si se pueden relacionar con los datos de OSM.

Fecha de fin: 01/02/2019

|  |  |
| --- | --- |
| Tarea | Estimación |
| Preprocesar las rutas | 13 |

## Sprint 5

Implementar un algoritmo de detección de paradas.

Fecha de fin: 15/02/2019

|  |  |
| --- | --- |
| Tarea | Estimación |
| Obtener stops en las rutas | 13 |

## Sprint 6

Implementar un segundo algoritmo de detección de paradas.

Fecha de fin: 01/03/2019

|  |  |
| --- | --- |
| Tarea | Estimación |
| implementación de un segundo algoritmo de detección de paradas | 5 |

## Sprint 7

El objetivo de este sprint es crear un primer modelo para los datos de la aplicación que son las trayectorias conceptuales y crear una primera base de datos que las contenga.

Fecha de fin: 14/03/2019

|  |  |
| --- | --- |
| Tarea | Estimación |
| Refactorizar código | 5 |
| Crear Modelo Conceptual | 5 |
| Crear Base de Datos | 8 |

### Sprint 8

Modificar el diseño de la base de datos del anterior sprint, empezar con la documentación e implementar funciones para cargar datos en la base de datos.

Fecha de fin: 29/03/2019

|  |  |
| --- | --- |
| Tarea | Estimación |
| Hacer documentación | 5 |
| Adaptar la base de datos a los cambios | 2 |
| Implementar la carga de datos en la base de datos | 8 |

### Sprint 9

Cargar los datos en la base de datos y ampliar la documentación.

Fecha de fin: 10/04/2019

|  |  |
| --- | --- |
| Tarea | Estimación |
| Memoria rellenar conceptos teóricos | 5 |
| Cargar datos de la BD | 3 |

### Sprint 10

Preparar un servidor con la aplicación de Nominatim para la busqueda de puntos de interés y estudiar un algoritmo de predicción.

Fecha de fin: 26/04/2019

|  |  |
| --- | --- |
| Tarea | Estimación |
| Manual de Instalación de Nominatim | 5 |
| Instalar Nominatim | 8 |
| Lectura de datos de Nominatim | 5 |
| Estudiar el algoritmo de predicción | 3 |

### Sprint 11

Probar algoritmos de *clustering* y el clasificador estudiado en el sprint anterior.

Fecha de fin: 17/05/2019

|  |  |
| --- | --- |
| Tarea | Estimación |
| Probar Clustering | 8 |
| Probar Clasificador de árbol | 8 |

### Sprint 12

Resolver problemas que se podían apreciar en el clasificador dado que había demasiadas clases para predecir y muy pocos datos.

Fecha de fin: 30/05/2019

|  |  |
| --- | --- |
| Tarea | Estimación |
| Crear arboles tipo | 5 |
| Agrupan clases por grupos | 13 |
| Resolver el problema de los edificios sin tipo | 13 |

### Sprint 13

Aprender como realizar aplicaciones web mas concretamente con el *framework* de Flask y crear un diseño base de la aplicación.

Fecha de fin: 03/06/2019

|  |  |
| --- | --- |
| Tarea | Estimación |
| Diseño Base de la aplicación web | **5** |
| Aprender Flask | 5 |

### Sprint 14

Avanzar a la memoria y los anexos y tratar de emplear un algoritmo de *clustering* utilizando matrices de adyacencia.

Fecha de fin: 11/06/2019

|  |  |
| --- | --- |
| Tarea | Estimación |
| Anexo B: B-1, B-2, B-3 | **8** |
| Anexo C: Diagramas de casos de uso | 5 |
| Crear un algoritmo de clustering | **8** |
| Memoria: Conceptos teóricos | 5 |
| Memoria: Objetivos | **3** |
| Memoria: Introducción | 3 |

### Sprint 15

Finalizar la memoria la pagina web y los anexos.

Fecha de fin: 02/07/2019

|  |  |
| --- | --- |
| Tarea | Estimación |
| Terminar la memoria | **8** |
| Terminar página web | 13 |

En la siguiente tabla tenemos las sumas de los tiempos de cada sprint y el total, se si sacamos la media de los tiempos nos da que cada sprint debería durar 18,2 fracciones de tiempo.

|  |  |
| --- | --- |
| Sprint 1 | 38 |
| Sprint 2 | 19 |
| Sprint 3 | 13 |
| Sprint 4 | 13 |
| Sprint 5 | 13 |
| Sprint 6 | 5 |
| Sprint 7 | 18 |
| Sprint 8 | 15 |
| Sprint 9 | 8 |
| Sprint 10 | 21 |
| Sprint 11 | 16 |
| Sprint 12 | 31 |
| Sprint 13 | 10 |
| Sprint 14 | 32 |
| Sprint 15 | 21 |
| Total | 273 |

## A.3. Estudio de viabilidad

### Viabilidad económica

En este apartado de viabilidad económica se va a a calcular los costes que tendría el desarrollo de esta aplicación para una empresa. Y los costes de explotación anuales de la aplicación plenamente operativa y con capacidad para trabajar con datos de todo el planeta. Los requisitos del sistema están basados en recomendaciones de Nominatim para manejar una base de datos de OSM completa.

#### Costes de desarrollo

Primero se calculan los costes que supone contratar a un desarrollador, para calcular esto se estima que un programador cobra 1500€ brutos y se estima el coste de la seguridad social en un 33%. También se tiene en cuenta que la duración del proyecto son 30 horas por crédito siendo 12 créditos. Esto hacen 360 horas que son aproximadamente dos meses a jornada completa.

|  |  |
| --- | --- |
| Sueldo bruto | 1.500 € |
| Seguridad social | **495€** |
| Total | **3.990 €** |

A esto hay que añadir los costes derivados del equipo necesario para el desarrollo del software. Se calcula un plazo de amortización de 5 años para los equipos informáticos. Partimos de que un portátil promedio para este fin tiene un coste de 800€ con sistema operativo incluido.

|  |  |
| --- | --- |
| Office 365 Empresa Premium | 21,00€ |
| Visual Studio Code | **0,00€** |
| Ordenador | **26,66€** |
| Total | **47,66€** |

Esto hace un coste total de desarrollo de **4.037,66€**

#### Costes de explotación

Para esto vamos a calcular el coste mensual que tiene un servidor Con 32GB de memoria RAM, 8 vCPU y una capacidad de 2TB de disco duro sólido. Los costes son mensuales, están estimados para la plataforma Azure de Microsoft y pueden variar sensiblemente.

|  |  |
| --- | --- |
| Servidor virtual | 240,93€ |
| 2 TB disco duro SSD | **198,62€** |
| Ubuntu server 18.04 | **0€** |
| Total | **439,55€** |

El coste anual de explotación se sitúa en **5.274,60€**

### Viabilidad legal

En este apartado se estudia la viabilidad legal de la aplicación. Para esto se tiene en cuenta en primer lugar las licencias del software integrado en nuestra aplicación.

|  |  |
| --- | --- |
| Pandas | BSD |
| GeoPandas | **BSD 3** |
| SQLAlchemy | **MIT** |
| Flask | **BSD** |
| Prefixspan | **MIT** |
| BootStrap | **MIT** |

Todas ellas pueden usarse en una aplicación con licencia AGPL3 por lo que no supone ningún problema. Esta licencia es similar a GPL3, pero está pensada para aplicaciones Web.

El otro punto problemático puede ser el tratamiento de datos, pero al no contener datos personales ni que permitan identificar a individuos no hay ningún problema legal. Siempre y cuando los datos sean obtenidos de forma legal, informado a los usuarios de su uso.

# Especificación de Requisitos

## B.1. Introducción

En este apéndice se exponen los requisitos que debe tener el proyecto. Primero se especifican los objetivos generales de la aplicación software y posteriormente se pasa a catalogar todos los requisitos de la aplicación. Por último, se realiza una especificación detallada de los requisitos software de la aplicación.

## B.2. Objetivos generales

Los requisitos de funcionamiento que va a tener este proyecto son los siguientes.

* El objetivo principal que se persigue en este proyecto es crear una aplicación que permita analizar trayectorias. Con este fin se van a desarrollar e implementar las siguientes características.
  + Guardar los datos de la aplicación en una base de datos, estos datos se guardarán ya procesados.
    - Se precisa eliminar errores que pueda haber en las trayectorias debido a fallos aleatorios o de GPS.
    - Detectar cuales, de los puntos de la ruta, un usuario está parado y agrupar los puntos consecutivos en los que se determina que el usuario está parado.
    - Asignar identificadores de OSM a las paradas, estos identificadores corresponden a la dirección en la que determina que está parado.
  + Permitir al usuario decidir exactamente qué datos de la base de datos de la aplicación quiere cargar en el sistema para usar las distintas funcionalidades de la aplicación.
  + Predictor que permita predecir a qué categoría o tipo pertenece la siguiente parada de una trayectoria.
  + Una interfaz amigable para que el usuario pueda utilizar la aplicación.
  + Permitir al usuario que pueda exportar datos de le la base de datos a un fichero CSV.

## B.3. Catálogo de requisitos

A continuación, se enumeran los requisitos funcionales y no funcionales de la aplicación.

### Requisitos funcionales

* **RF1 - Cargar datos CSV en la aplicación**: El usuario debe poder agregar nuevos datos procedentes de ficheros CSV.
  + **RF1.1 – Espera en carga de datos:** El usuario debe ser consciente de que los datos se están cargando en la aplicación y debe esperar.
  + **RF1.2 – Resultado de la carga de datos**: El usuario debe recibir información referente a los datos que se han cargado en la aplicación tras el proceso de carga.
  + **RF1.3 – Exportar resultados de carga:** El usuario podrá exportar los resultados que recibe una vez terminado el proceso de carga de datos.
* **RF2 – Predecir clases de las rutas:** El usuario deberá poder entrenar un clasificador con un conjunto de datos seleccionado, para predecir la clase a la que pertenecerá el siguiente punto de las rutas.
  + **RF2.1 – Selección de datos:** El usuario debe poder elegir que datos desea utilizaren cada momento.
  + **RDF2.2 – Validación cruzada:** El usuario visualizara los resultados de realizar una validación cruzada con el conjunto de datos cargado antes de entrenar el clasificador.
  + **RF2.3 – Datos insuficientes:** El usuario será informado en caso de haber datos insuficientes para entrenar un clasificador, en cuyo caso no podrá entrenar el clasificador con esos datos.
  + **RF2.4 – Reiniciar clasificador:** El usuario podrá reiniciar el clasificador a si como todos los datos que están cargado en la aplicación para ser utilizados.
  + **RF2.5 – Múltiples predicciones:** El usuario podrá utilizar el clasificador múltiples veces para predecir Trayectorias semánticas
  + **RF2.6 – Exportar resultado de predicción:** El usuario podrá exportar los resultados de la predicción obtenidos.
* **RF3 – Exportar datos de la aplicación:** El usuario deberá poder exportar los datos contenidos en la aplicación mediante una sentencia de tipo select.
  + **RF3.1 – Respuesta al usuario:** El usuario debe recibir un aviso si la sentencia select no está bien formulada o un fichero CSV si la operación fue procesada correctamente.
* **RF4 – Datos de OSM:** El administrador podrá añadir nuevos datos de OSM a la aplicación de Nominatim.

### Requisitos no funcionales

**RNF1 – Facilidad de uso:** El usuario debe recibir en todo momento información sobre los fallos en el uso de la aplicación y el estado de la aplicación

**RNF2 – Información adicional:** El usuario recibirá información adicional en el contexto de la aplicación o facilidades para no necesitar conocer la estructura de la base de datos para poder usarla.

### Limitaciones Funcionales

**LF1 – Usuarios:** La aplicación no deberá ser utilizada por mas de un usuario al mismo tiempo ya que compartirían la misma sesión.

## B.4. Especificación de requisitos

En este apartado se van a especificar los casos de uso, que requisitos funcionales va a abarcar y que característica tiene cada caso de uso.

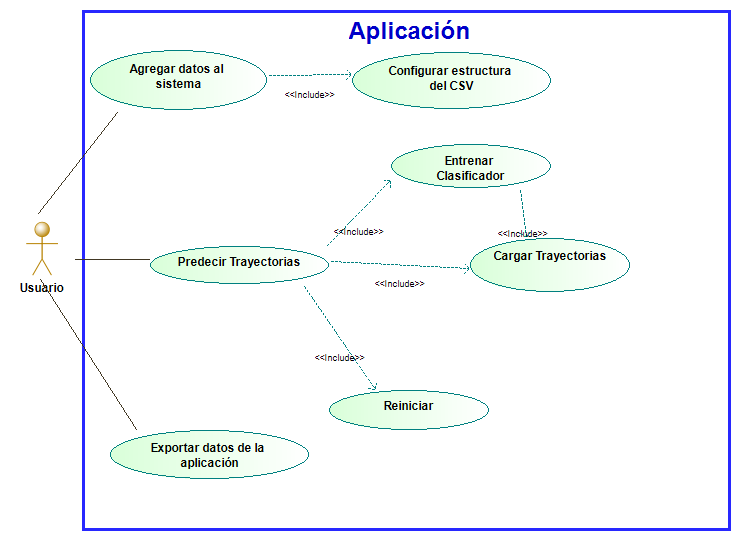
### Actores

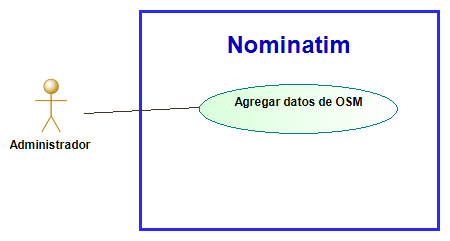
En los casos de uso que se exponen a continuación encontramos dos actores con roles completamente distintos.

**Usuario:** Este es el actor principal de la aplicación y es además el único que interactúa directamente con ella mediante la página web.

**Administrador:** Este es el encargado de que en la aplicación los usuarios tengan los datos necesarios para que Nominatim pueda asociar puntos geográficos con puntos de interés.

### Diagramas de casos de uso





### Especificación de los casos de uso

|  |  |  |
| --- | --- | --- |
| Nombre | CU1 – Agregar datos al sistema | |
| Descripción | El usuario agrega datos al sistema | |
| Requisitos funcionales | RF1, RF1.1, RF1.2, RF1.3 | |
| Usuario | Usuario | |
| Precondiciones | En la página web estar en la ventana de carga, haber rellenado correctamente la configuración del CSV | |
| Secuencia Normal | Paso | Acción |
| 1 | El usuario solicita cargar los datos |
| 2 | El navegador carga los datos en el servidor |
| 3 | El navegador pide la página cada 5 segundos al servido, si termina el proceso de carga, varia la página y salta al paso 7 |
| 4 | El servidor lee los archivos con extensión especificada |
| 5 | El servidor procesa los datos |
| 6 | El servido guarda los resultados en la base de datos |
| 7 | Se muestra la información de los resultados y habilita un botón para que el usuario pueda exportar el resultado. |
| Excepciones | Paso | Acción |
| 1 | Se produce un error al intentar cargar los datos |
| 2 | El servidor envía una ventana de error al navegador |
| Postcondiciones | Se añaden datos a la base de datos | |
| Frecuencia | Baja | |
| Importancia | Critica | |

|  |  |  |
| --- | --- | --- |
| Nombre | CU1.1 – Configurar estructura del CSV | |
| Descripción | El usuario configura el lector de CSV | |
| Requisitos funcionales | RF1 | |
| Usuario | Usuario | |
| Precondiciones | Estar en la ventana de carga de datos | |
| Secuencia Normal | Paso | Acción |
| 1 | El usuario rellena los campos |
| 2 | El usuario pulsara el botón de cargar |
| Excepciones | Paso | Acción |
| 1 | Al pulsar agregar hay campos obligatorios sin rellenar o campos inválidos |
| 2 | El navegador informa al usuario del primer campo vacío o invalido |
| Postcondiciones | Permite cargar los datos en el servidor | |
| Frecuencia | Baja | |
| Importancia | Critica | |

|  |  |  |
| --- | --- | --- |
| Nombre | CU2 – Predecir Trayectorias | |
| Descripción | Permite al usuario predecir la clase a la que pertenecerá la próxima ubicación en una trayectoria | |
| Requisitos funcionales | RF2, RF2.5, RF2.6 | |
| Usuario | Usuario | |
| Precondiciones | Tener el clasificador entrenado, tener datos cargados | |
| Secuencia Normal | Paso | Acción |
| 1 | El usuario pulsa el botón de predecir |
| 2 | El servidor pasa los datos actualmente cargados por el clasificador |
|  | 3 | El navegador muestra en una tabla los resultados y permite exportarla |
|  | 4 | El usuario puede exportar la tabla y/o volver a predecir o cargar datos |
| Excepciones | Paso | Acción |
|  |  |
| Postcondiciones |  | |
| Frecuencia | Alta | |
| Importancia | Critica | |

|  |  |  |
| --- | --- | --- |
| Nombre | CU2.1 – Entrenar clasificador | |
| Descripción | Entrena de forma automática un clasificador, realiza la validación cruzada | |
| Requisitos funcionales | RF2, RF2.5 | |
| Usuario | Usuario | |
| Precondiciones | Terminar de cargar los datos en el sistema, no hay clasificador entrenado | |
| Secuencia Normal | Paso | Acción |
| 1 | El servidor realiza una validación cruzada |
| 2 | Entrena un clasificador |
| 3 | Presenta los resultados de la validación en el navegador |
| Excepciones | Paso | Acción |
|  |  |
| Postcondiciones | Obtenemos un clasificador entrenado | |
| Frecuencia | Alta | |
| Importancia | Critica | |

|  |  |  |
| --- | --- | --- |
| Nombre | CU2.2 - Reiniciar | |
| Descripción | Devuelve a todos los componentes de predicción a su estado inicial | |
| Requisitos funcionales | RF2.4 | |
| Usuario | Usuario | |
| Precondiciones | Estar en la ventana de predicciones | |
| Secuencia Normal | Paso | Acción |
| 1 | El usuario pulsa el botón de reiniciar |
| 2 | El servido devuelve las variables de predicción a su estado original |
| 3 | Se refresca la página web |
| Excepciones | Paso | Acción |
|  |  |
| Postcondiciones | Desaparecen todo lo realizado en la ventana predicciones | |
| Frecuencia | Media | |
| Importancia | Alta | |

|  |  |  |
| --- | --- | --- |
| Nombre | CU2.3 – Cargar Trayectorias | |
| Descripción | Carga trayectorias en memoria | |
| Requisitos funcionales | RF2.1, RF2.3 | |
| Usuario | Usuario | |
| Precondiciones | Estar en la ventana de predicción | |
| Secuencia Normal | Paso | Acción |
| 1 | El usuario rellena o no los campos de la columna cargar datos |
| 2 | El usuario presiona el botón de cargar |
|  | 3 | Se envía una solicitud al servidor |
|  | 4 | El servidor almacena los datos en memoria |
| Excepciones | Paso | Acción |
| 1 | Si hay campos incompatibles con la consulta select devuelve un error |
| Postcondiciones | Datos cargados en el sistema | |
| Frecuencia | Alta | |
| Importancia | Critica | |

|  |  |  |
| --- | --- | --- |
| Nombre | CU3 – Exportar datos de la aplicación | |
| Descripción | Permite a un usuario exportar datos almacenados en la base de datos | |
| Requisitos funcionales | RF3, RF3.1 | |
| Usuario | Usuario | |
| Precondiciones | Estar en la ventana exportar, que el usuario tenga conocimientos de SQL básicos | |
| Secuencia Normal | Paso | Acción |
| 1 | El usuario rellena la consulta select |
|  | 2 | El usuario presiona el botón para hacer la exportación |
|  | 3 | El servidor detecta si la consulta pide datos geoespaciales, en caso afirmativo ir al paso 5 |
|  | 4 | El servidor hace una petición normal a la base de datos saltar al paso 7 |
|  | 5 | El servidor realiza una consulta con datos geoespaciales |
|  | 6 | El servidor convierte la columna de datos geoespacial en una de longitud y otra de latitud |
|  | 7 | Se devuelve un archivo CSV con los datos |
| Excepciones | Paso | Acción |
| 1 | Si la consulta es invalida el servidor devuelve una notificación |
| Postcondiciones |  | |
| Frecuencia | Baja | |
| Importancia | Media | |

|  |  |  |
| --- | --- | --- |
| Nombre | CU4 – Agregar datos de OSM | |
| Descripción | Añadir datos a Nominatim | |
| Requisitos funcionales | RF4 | |
| Usuario | Administrador | |
| Precondiciones | Estar logrado con el usuario de Nominatim en el servidor, tener internet | |
| Secuencia Normal | Paso | Acción |
| 1 | El administrador debe descargar los nuevos datos de internet |
| 2 | El administrador debe pasar estos datos al setup de Nominatim |
| Excepciones | Paso | Acción |
|  |  |
| Postcondiciones | Nuevas zonas para obtener puntos de interés | |
| Frecuencia | Baja | |
| Importancia | Critica | |

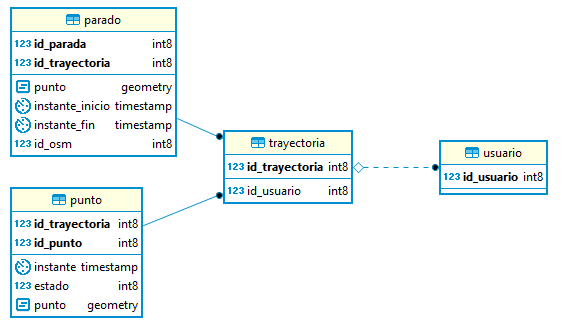
# Especificación de diseño

## C.1. Introducción

En este apartado se exponen los aspectos de diseño más importantes de la aplicación. En el apartado de diseño de datos se define como están estructurados los datos y como se almacena. En el diseño arquitectónico se explica cómo está estructurada la aplicación. Por último, en el diseño procedimental explican aquellas características más importantes de los módulos que conforman la aplicación.

## C.2. Diseño de datos

En este proyecto se utilizan bases de datos concretamente bases de datos geoespaciales. Primero vamos a analizar el diseño de la base de datos que aparece en la siguiente ilustración.



Vamos a analizar el diagrama de derecha a izquierda y de arriba abajo. Lo primero que encontramos es la tabla usuario la cual solo tiene el id de usuario. La razón por la que se creó esta tabla en vez de integrarse en las trayectorias se debe a que se barajaba la posibilidad de guardar información sobre el usuario después de un proceso de análisis de sus trayectorias. Tiene una relación de “1 a N” con las trayectorias dado que un usuario tiene muchas trayectorias.

La tabla de trayectoria fundamentalmente relaciona a los usuarios las trayectorias tanto las formadas por paradas (que también denominamos conceptuales) y los formadas por puntos (que denominamos trayectorias conceptuales). Con ambas tablas tiene una relación de “1 a N” dado que una trayectoria bruta está formada por “N” puntos y una trayectoria conceptual está formada por “N” paradas.

La tabla, parado está formado por una clave doble por un lado la de la trayectoria que se repite en todas las paradas de la misma trayectoria y por “id\_parada” que enumera a las paradas de una mima trayectoria. Esto nos permite identificarlas de forma única y además podemos mantener su correcto orden. En la tabla punto funciona de la misma manera. En la tabla parado se almacenan dos valores de tiempo que corresponden a el tiempo del inicio de la parada y a el tiempo del fin de la parada. El campo “id\_osm” se utiliza para buscar los datos del punto de interés relacionado con esa parada. Ya que ese campo es el del identificador en OSM del punto de interés relacionado con la parada.

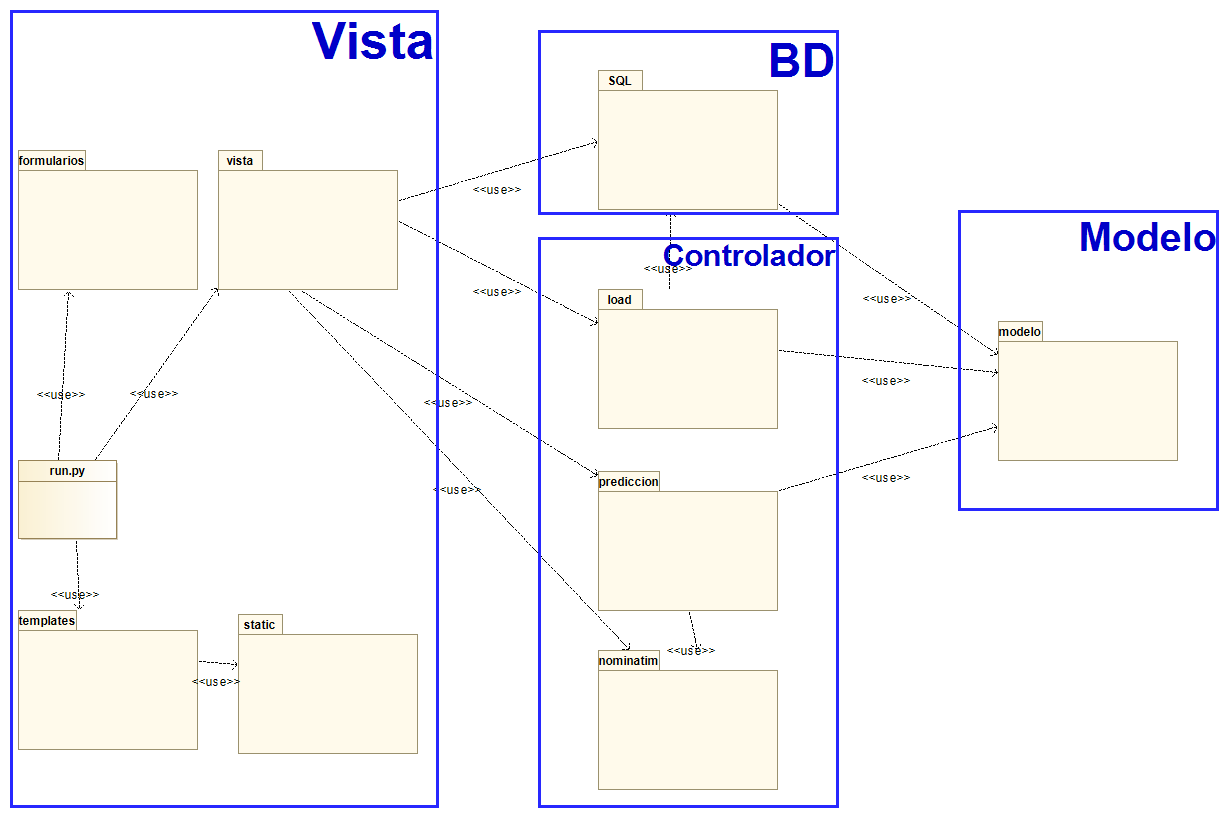
Por último, ambas tablas tienen un campo punto de tipo geometría el cual contiene en cierto modo las coordenadas en las que se realizó la parada o en punto por el que paso el usuario. Realmente no son las coordenadas si no un valor que utiliza PostGIS para identificar al punto en sus tablas propias.

En la aplicación cada trayectoria se almacena en un objeto la cual contiene *DataFrames* con estos datos, más concretamente *GeoDataFrames así como la id de usuario y la id de la trayectoria.*

Las trayectorias semánticas en la aplicación son más un concepto que una realidad, ya que para no tener información redundante se obtiene a través de las trayectorias conceptuales. Se agrupan las paradas por id de OSM y se obtiene la información que buscamos.

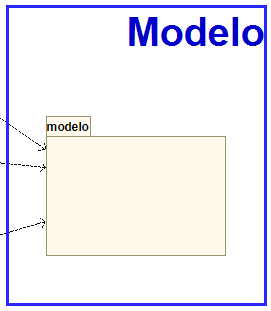
## C.3. Diseño arquitectónico

Se describe como está distribuida la aplicación y como se relaciona sus componentes. En la ilustración siguiente se observa un plano general de cómo se agrupan los componentes en el patrón de diseño MVC. Posteriormente se verá más de cerca cada uno de sus componentes y que se encapsula en estos.



### Modelo

El modelo en este diseño solo se accede de forma directa por el controlador y el módulo de bases de datos. Al tratarse de una aplicación que se centra en un modelo de datos concreto estos están agrupados en un solo modulo.



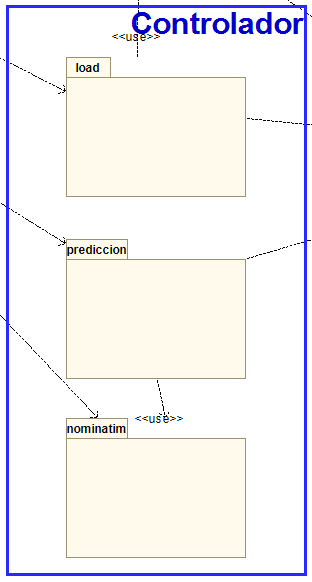
### Controlador

Este grupo está formado por tres componentes los cuales cada uno encapsula una parte de la lógica del programa.

Load: Se encarga de cargar los datos de los archivos y convertirlos en trayectorias. A este solo se accede desde la vista cuando un usuario quiere introducir nuevas trayectorias en el sistema. A su vez este accede a las bases de datos para saber que claves tiene que asignar a usuarios y trayectorias. También accede al modelo de datos dado que lo que hace es construir trayectorias brutas y conceptuales en base a este.

Predicción: Se encarga de toda la lógica del programa relacionada con minería de datos propiamente dicha. A esta se accede desde el módulo de vista, y esta a su vez accede a el modelo y al módulo de Nominatim.

Nominatim: Este módulo bien se podría haberse situado dentro de bases de datos, aunque no lo es en el sentido estricto, pero tiene una finalidad similar. Este módulo no necesita de ningún otro ya que solo se encarga de la comunicación entre la aplicación y el servidor de Nominatim. Este módulo recibe peticiones tanto de la vista como del módulo de predicciones.



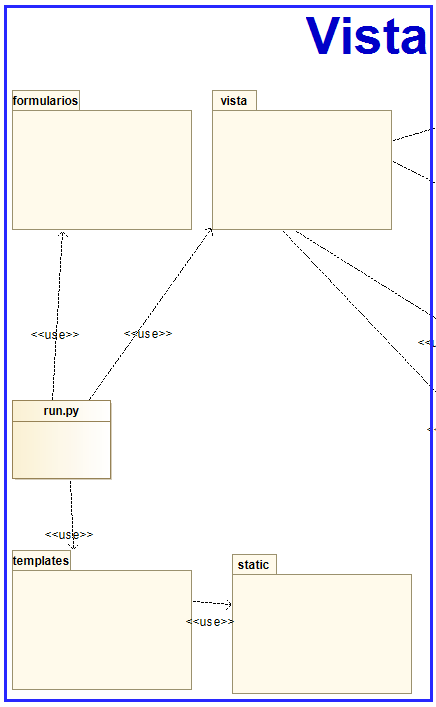
### Vista

Este grupo fundamentalmente representa en este caso a la parte de web más concretamente a Flask. Flask nos obliga a tener para su propio funcionamiento templates y static los cuales no contiene código Python. Templates contiene las paginas HTML y static contiene los archivos de estilos CSS y los archivos de Java Script.

Formularios: contiene los formularios que se utilizaran en la página web por parte de Flask. Están en código Python, pero solo se relaciona con “run.py” para que Jinja2 los pueda renderizar.

Vista: Contiene clases utilizadas por la vista para almacenar información que necesita y la clase que agrupa las funciones que se necesitan del modelo y de la base de datos. Es el único módulo de la vista que se comunica con el resto de la aplicación.

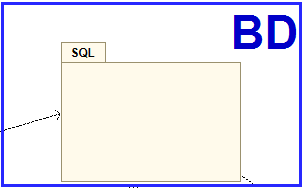
Run.py: Esta clase es la clase principal de Flask ya que es la que arranca el servidor de la página web y es donde se definen las solicitudes HTTP que aceptara el servidor y que respuesta se da a esas solicitudes.



### Base de Datos

El módulo de bases de datos se encarga de todo en tráfico de datos entre la base de datos y la aplicación. Está encargado de establecer conexiones, de convertir el modelo a registros de la base de datos y de convertir los registros de la base de datos en el modelo.

Este módulo es utilizado tanto por parte de la vista como por parte del módulo de load. Este a su vez depende del modelo para devolver los datos estructurados en base a este.



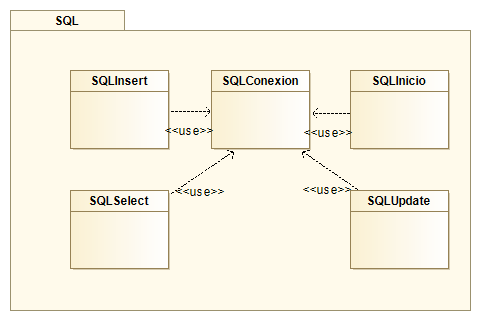
## C.4. Diseño procedimental

En este apartado se tratan un poco más en profundidad como se relaciona algunos de los componentes de la aplicación. Como algunos de ellos son muy básicos y fáciles de entender no se van a detallar y se pondrá el foco en aquellos que puedan ser más difíciles de comprender.

Se debe destacar que esta no es una aplicación orientada a objetos, aunque si se utilizan. Esto implica que no todos los archivos de la aplicación contienen clases, si no que por el contrario algunos actúan como ficheros de configuración o como agrupaciones de funciones. Un buen ejemplo de esto es el paquete SQL el cual no tiene ninguna clase y los ficheros están compuestos por agrupaciones de funciones.

### SQL

En la siguiente ilustración (que muestra un diagrama de clases, aunque no son clases) muestra cómo se relacionan y el funcionamiento es el mismo para todos los ficheros. El fichero “SQLConexion” guarda los datos de configuración del servidor y además tiene una función que proporciona una conexión con ese a quien la invoca. El único fichero que tiene un comportamiento distinto es “SQLInicio” que proporciona un contador actualizado al módulo “load” para asignar las ID a usuarios y trayectorias.

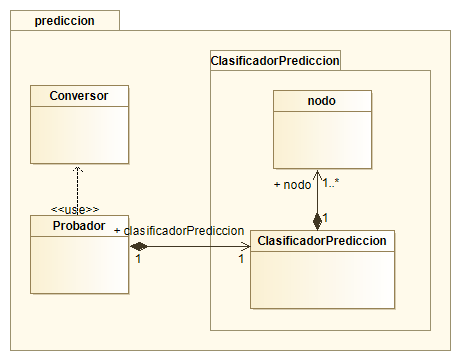


### Predicción

Conversor: Esta clase tiene distintas herramientas de conversión para convertir los datos de trayectorias a otros formatos que pueden entender clasificadores y algoritmos de Clustering.

Probador: Es una clase construida con el objetivo de probar el clasificador incluye validación cruzada y cálculo de distintas medidas para evaluar la calidad del clasificador.

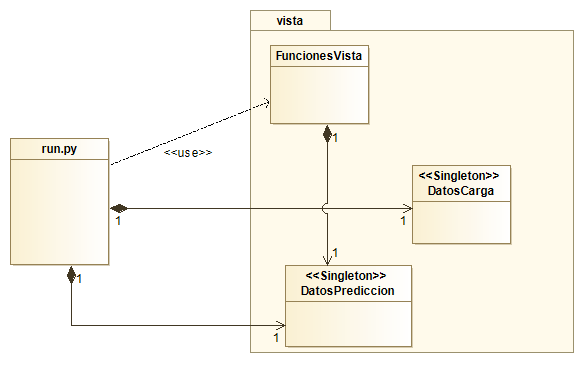
ClasificadorPrediccion: Es al mismo tiempo clase y paquete ya que alberga en el mismo fichero una clase nodo la cual utiliza para construir sus árboles.



### Vista

En la vista hay que destacar las comunicaciones entre “run” y “FuncionesVista” estas funciones tiene una serie de operaciones que con grandes volúmenes de datos pueden suponer un tiempo de ejecución excesivamente lago. Como el navegador no puede esperar mucho tiempo a que “run” responda lo que se hace es ejecutar procesos o hilos separados. Estos se almacenan en las dos clases *sigleton* que aparecen en la ilustración. Lo que se hace “run” es lanzar la función de cargar como un proceso y lo almacena en la clase “DatosCargar” a si puede saber en todo momento si este proceso ha terminado, ya que pregunta en otras fases de ejecución. Esto es necesario debido a que run solo ejecuta alguna de sus funciones cuando el navegador hace alguna petición al servido. Como “run” debe responder en un corto periodo de tiempo perdiendo el control no puede preguntar una y otra vez sin perder las variables de la función. Tampoco puede guardar variables globales porque no es una clase, por lo que una clase *singleton* puede solucionar este problema.

En el caso de “DatosPrediccion” se utiliza un hilo en vez de un proceso realizando una tarea similar con la diferencia que ahora este hilo también guarda datos en la clase “DatosPrediccion”.



# Documentación técnica de programación

## D.1.Introducción

En este apartado se van a exponer distintas características técnicas del proyecto. Primero se expondrá la estructura de directorios del proyecto. Posteriormente se expondrán los pormenores relacionados con las necesidades para continuar desarrollando el software. A continuación, se explica cómo se ha de instalar el software y como se debe ejecutar. Por último, se expondrán las pruebas a las que ha sido sometida la aplicación.

## D.2. Estructura de directorios

## D.3. Manual del programador

## D.4. Compilación, instalación y ejecución del proyecto

## D.5. Pruebas del sistema

# Documentación de usuario

## E.1. Introducción

## E.2. Requisitos de usuarios

## E.3. Instalación

## E.4. Manual del usuario