

**Universidad EAFIT**

**Escuela de Ingeniería**

**Especialización en Desarrollo de Software**

**Arquitectura de Software**

**Arquitectura del Sistema de Información:**

**Control de Transporte público**

**Aplicación Quick Bus**

**ARC42**

Presentado a

*José Mauricio Álvarez H.*

Elaborado por

Gersaín David Castañeda Muñoz

Andrés Felipe Erazo Ramos

*Martin Julián Osorio Marulanda*

*Duván Arlen Corrales Marín*

*Medellín - 2019*

**Plantilla** 

2018-12-20

# 

**Acerca de arc42**

arc42, La plantilla de documentación para arquitectura de sistemas y de software.

Por Dr. Gernot Starke, Dr. Peter Hruschka y otros contribuyentes.

Revisión de la plantilla: 7.0 ES (basada en asciidoc), Enero 2017

© Reconocemos que este documento utiliza material de la plantilla de arquitectura arc 42, [http://www.arc42.de](http://www.arc42.de/). Creada por Dr. Peter Hruschka y Dr. Gernot Starke.

Contenido

[1. Introducción y Metas 4](#_Toc25000377)

[1.1. Resumen de requisitos 4](#_Toc25000378)

[1.2. Uso básico 5](#_Toc25000379)

[1.3. Funcionalidad general 5](#_Toc25000380)

[2. Metas de calidad 7](#_Toc25000381)

[3. Stakeholders 7](#_Toc25000382)

[4. Restricciones de arquitectura 9](#_Toc25000383)

[5. Contexto 10](#_Toc25000384)

[6. Estrategia de solución 16](#_Toc25000385)

[7. Vista de despliegue o implementación 18](#_Toc25000386)

[8. Conceptos transversales (Cross-cutting) 21](#_Toc25000387)

[8.1. Modelos de Dominio 21](#_Toc25000388)

[8.2. Persistencia 22](#_Toc25000389)

[8.3. Interfaz de Usuario 22](#_Toc25000390)

[8.4. Comunicaciones e Integración 23](#_Toc25000391)

[8.5. Excepción y Manejo de Errores 23](#_Toc25000392)

[8.6. Logging, Tracing 23](#_Toc25000393)

[8.7. Configurabilidad 23](#_Toc25000394)

[8.8. Testability 24](#_Toc25000395)

[8.9. Internationalizacion 24](#_Toc25000396)

[8.10. Build-Management 24](#_Toc25000397)

[9. Decisiones de diseño 24](#_Toc25000398)

[10. Escenarios de calidad 26](#_Toc25000399)

[11. Riesgos y deuda técnica 28](#_Toc25000400)

[12. Glosario 28](#_Toc25000401)

[13. Anexos 30](#_Toc25000402)

[14. Bibliografía 34](#_Toc25000403)

# Introducción y Metas

La ciudad de Medellín, de aproximadamente 2.5 Millones de habitantes, siempre se ha caracterizado por su cultura ciudadana y por este motivo el gobierno local ha decidido implementar un sistema para tener mayor control el uso del transporte público, además de proveer a los ciudadanos una aplicación móvil que les será de utilidad como una especie de mapa de ciudad, en la cual podrán obtener sugerencias de rutas para ir desde un origen hasta un destino usando los medios de transporte nativos incluyendo Encicla. También los ciudadanos podrán obtener información del clima para el punto de partida y llegada de las opciones de ruta.

# Resumen de requisitos

El gobierno de la ciudad tiene el objetivo de incrementar el acceso a la telefonía móvil desde el 75% hasta el 90% de la población en los próximos años, basado en esto también pretende ofrecer a sus habitantes una solución tecnológica denominada QUICK BUS, de tal manera que se optimicen los tiempos que se deben invertir al uso de transporte en bus. El objetivo es que el usuario del sistema fácilmente conozca las rutas disponibles para alcanzar un destino Y desde un origen X, identificar la ruta óptima, conocer en cuanto tiempo pasa el próximo bus, de tal manera que el usuario del sistema deba invertir el menor tiempo posible en sus desplazamientos y en cualquier momento conozca la información requerida disponible para tomar la decisión apropiada frente al tema del transporte público en bus.

Lo novedoso en este sistema de información es que se plantea integrar todos los medios de transporte nativos disponibles, así, para el valle de aburrá se tendría integración y cálculo de rutas con los medios de transporte tranvía, metroplus, metro, encicla, metrocable, integrados y alimentadores.

El hecho de tener claridad en la información y monitoreo permanente frente al recorrido de los desplazamientos en bus, le permitirán a la alcaldía de la ciudad efectuar análisis profundos sobre la modificación de recorridos, implementación de nuevas rutas. En general, la alcaldía espera que la información obtenida del sistema de control del transporte público en bus se convierta en materia prima fundamental que constituya una variable esencial para tomar decisiones relativas al transporte de la ciudad.

En un horizonte de largo plazo, se pretende que el sistema de información calcule la huella de carbono asociada a cada opción de ruta y que a su vez haga una comparación de la huella de carbono que se produciría si el ciudadano estuviese usando su carro.

# Uso básico

* 1. Un pasajero registrado en el sistema accede a la opción de consulta de rutas**.** El sistema presenta como filtros de búsqueda: Estación origen, estación destino y duración de recorrido.
  2. El pasajero visualiza la mejor ruta. El sistema utiliza los servicios de GPS y calcula cual es la ruta en la que se toma menor tiempo llegar desde el punto origen al destino. Finalmente se presenta en pantalla la mejor ruta.
  3. El pasajero visualiza el tiempo en el que llegará el próximo bus. El sistema según la estación en la que se encuentre el pasajero consulta los buses que tengan rutas que pasen por dicha estación y analiza el tiempo que estos van a tardar en llegar, tiempo que finalmente se presenta al usuario.
  4. El pasajero es alertado de que llegará el bus próximamente. El sistema valida que el tiempo de llegada del bus se encuentra dentro del parametrizado previamente y presenta la alerta al pasajero indicando que próximamente llegará el bus.
  5. La empresa transportadora registra un bus. El sistema presenta el formulario con los datos a registrar: Placa, capacidad, modelo, marca, documento y ruta asociada (desde menú desplegable del maestro de rutas).
  6. La empresa transportadora registra una ruta. El sistema presenta el formulario con los datos a registrar: Nombre, punto origen y punto destino de una ruta determinada.
  7. La empresa transportadora o la administración municipal visualizan reportes. El sistema presenta reportes según los parámetros de búsqueda ingresados.

# Funcionalidad general

Tabla de requisitos Generales

|  |  |  |
| --- | --- | --- |
| **ID** | **Funcionalidad** | **Descripción** |
| G-1 | Consultar rutas | El pasajero accede a la opción de consulta de rutas. El sistema presenta como filtros de búsqueda: Estación origen, estación destino y duración de recorrido, después de establecer los criterios de búsqueda, el sistema consulta las rutas que cumplan con dichos criterios.   El pasajero visualiza las rutas disponibles. |
| G-2 | Visualizar la ruta óptima | El sistema utiliza los servicios de GPS y calcula cual es la ruta en la que se toma menor tiempo llegar desde el punto origen al destino determinado por el pasajero. Finalmente se presenta en pantalla la mejor ruta. |
| G-3 | Presentar tiempo próximo bus | El sistema toma la ubicación actual del pasajero vía GPS e identifica en cual estación se encuentra. El sistema según la estación en la que se encuentre el pasajero consulta los buses que tengan rutas que pasen por dicha estación y analiza el tiempo que estos van a tardar en llegar, tiempo que finalmente se presenta al usuario. |
| G-4 | Alertar llegada de bus | El sistema valida el tiempo de llegada del bus que se encuentra dentro de los parámetros establecidos previamente por el pasajero y le notifica con una alerta indicando que próximamente llegará el bus dentro del tiempo estimado. |
| G-5 | Registro de buses | La empresa transportadora tiene la posibilidad de registrar un bus. El sistema presenta el formulario con los datos a registrar: Placa, capacidad, modelo, marca, documento y ruta asociada. |
| G-6 | Registro de rutas | La empresa transportadora ingresa a la opción “Registrar ruta”. El sistema presenta el formulario con los datos a registrar: Nombre, punto origen y punto destino. |
| G-7 | Generar reportes | Las empresas transportadoras y la administración municipal podrán generar reportes concernientes a los viajes en bus realizados. El sistema presenta como parámetros de búsqueda fecha de viaje y rutas y permitirá visualizar los reportes extraídos según los parámetros establecidos. |
| G-8 | Visualizar estado del clima | Visualizar un indicio del clima actual en el punto de partida y un clima aproximado según la hora y punto de llegada. |

# Metas de calidad

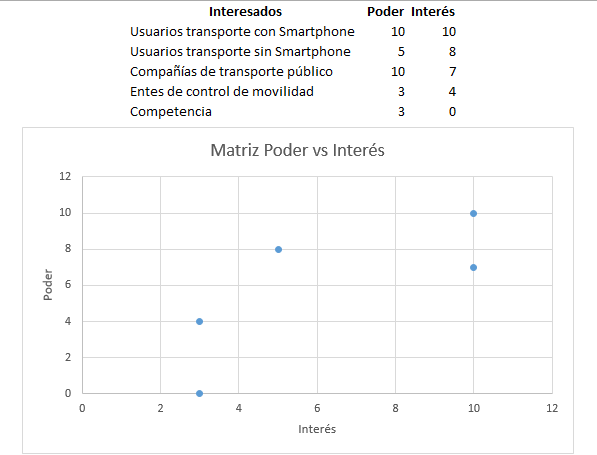
Tabla de objetivos de calidad

|  |  |  |
| --- | --- | --- |
| **Prioridad** | **Objetivo de Calidad** | **Guion** |
| 5 | Usabilidad | El sistema deber ser usable para los usuarios de forma que no sea necesario visitar más de 2 pantallas para ejecutar una funcionalidad disponible |
| 5 | Eficiencia | La plataforma debe hacer uso de los recursos, servicios y funcionalidades que ofrece un teléfono inteligente estándar, tales como: GPS, acceso a internet, alertas. |
| 5 | Portabilidad | El sistema para desarrollar debe ser móvil, para implementarse en teléfonos inteligentes (Smartphones) de sistema operativo Android y iOS. |
| 4 | Rendimiento | El sistema tendrá un óptimo rendimiento soportando la concurrencia de 10.000 usuarios. |
| 4 | Usabilidad | El sistema debe ser lo suficientemente simple para que un usuario que solo sepa leer y escribir pueda aprender a utilizarlo con una capacitación básica |
| 4 | Portabilidad | El sistema debe ser altamente parametrizable. |
| 3 | Disponibilidad | El sistema debe permanecer disponible todos los días del año desde las 4 am hasta las 00:00 cuando culmina la operación de los buses |
| 3 | Seguridad | Garantizar la confidencialidad de los datos que registran los pasajeros y las empresas de transportes. |

# Stakeholders

A continuación, se listan los interesados que interactúan o tienen alguna relación con el sistema.

|  |  |  |
| --- | --- | --- |
| **Rol** | **Descripción** | **Meta/Límite** |
| Usuario de transporte público de Medellín. | Habitantes de Medellín que tienen acceso a smartphone y hacen uso del transporte público. | Usuarios que entrarán en contacto directo a la aplicación, harán uso de sus funcionalidades para facilitar sus desplazamientos en transporte público. |
| Usuarios de transporte público sin smartphone. | Usuarios que en un futuro próximo utilizarán la aplicación QuickBus. | Según el estudio de la localidad de Medellín se espera un crecimiento de usuarios con smartphone en los próximos años, por lo tanto son usuarios potenciales de la aplicación. |
| Compañías de transporte público. | Compañías que prestan el servicio de transporte público en Medellín. | Las compañías de transporte son el principal aliado en la creación de la aplicación QuickBus, estas prestarán su servicio de transporte que será abstraído para ser la base de las funcionalidades de las que se valdrán los usuarios. |
| Entes de control de movilidad. | Entidades que presentan las normas a cumplir por las compañías de transporte. | Entidades que presentan la norma con la que deben cumplir los sistemas o aplicaciones móviles que ayudan a la gestión de la movilidad; según las normas que existan se pueden presentar restricciones a tener en cuenta. |
| Gobierno de la ciudad de Medellín. | Entidad encargada de velar por el crecimiento de la ciudad de Medellín y el bienestar de sus habitantes. | Con la información recolectada pretende tomar decisiones para mejorar la prestación del servicio de transporte público en bus y por ende aportar al bienestar de los habitantes. |
| Competencia | Aplicaciones o plataformas que también optimizan desplazamiento en transporte de personas. | Aplicaciones o plataformas que pueden ser base para validar funcionalidades propias o debilidades. |



# Restricciones de arquitectura

Restricciones técnicas.

Para el desarrollo y la implementación de la aplicación Quick Bus se deben tener en cuenta las siguientes restricciones:

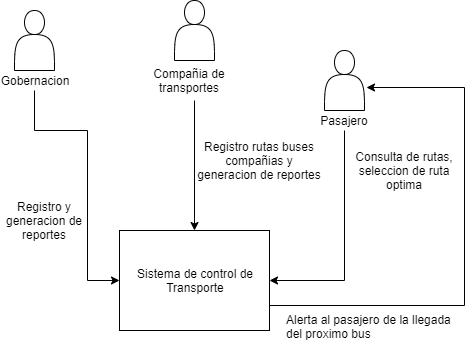
* + El sistema debe ser móvil, para implementarse en teléfonos inteligentes (Smartphones) de sistema operativo Android y iOS.
  + La plataforma debe servirse y apoyarse en los servicios y funcionalidades que ofrece un teléfono inteligente estándar, tales como: GPS, acceso a internet, alertas.
  + Se requiere la implementación de microservicios que permitan satisfacer uno o varios de los requisitos, de tal manera que se mejore la mantenibilidad y escalabilidad del software.
  + Es importante que la aplicación a desarrollar sea diseñada para implementarse bajo un modelo de ejecución en la nube, de tal manera que el equipo de desarrollo se dedique a las tareas propias de la construcción del proyecto y se desligue de las actividades requeridas por los temas de infraestructura. Uno de los objetivos que se persigue es tener flexibilidad en la escalabilidad horizontal para los ambientes de ejecución.

Restricciones organizacionales.

* + Por el corto tiempo definido según el cronograma establecido para la asignatura se procede a desplegar la aplicación en varios realeases donde inicialmente se entregan las funcionalidades de cálculo de ruta más optima y el pronóstico del clima.

# Contexto

C4 Model: Nivel 0 Contexto



Los pasajeros o usuarios realizan consulta de opciones de transporte para desplazarse desde un lugar de origen hasta un lugar de destino.

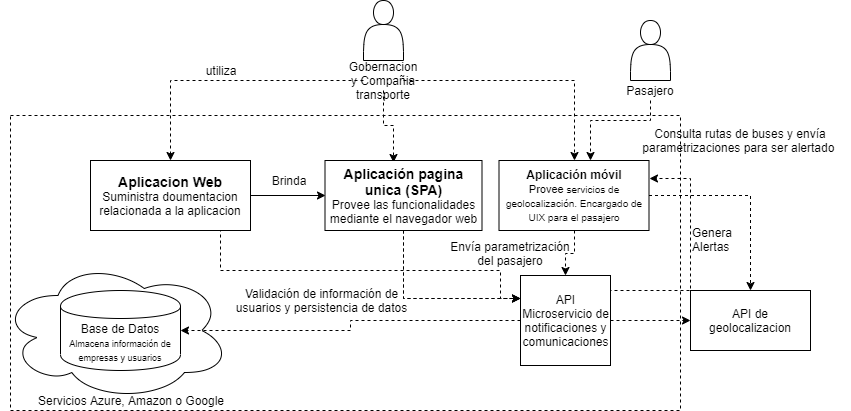
El sistema de control de transporte responde al usuario con opciones de ruta, teniendo en cuenta los medios de transporte nativos disponibles que pueden intervenir dados los puntos de origen y destino.

El sistema de control de transporte brinda al usuario información del clima en el punto de origen y hora actual, así como en el punto de destino y hora estimada de destino de la ruta que el usuario ingresó.

La Gobernación o entidad administradora del sistema, construye y genera reportes dinámicos usando una interfaz para reportes que provee el sistema de información.

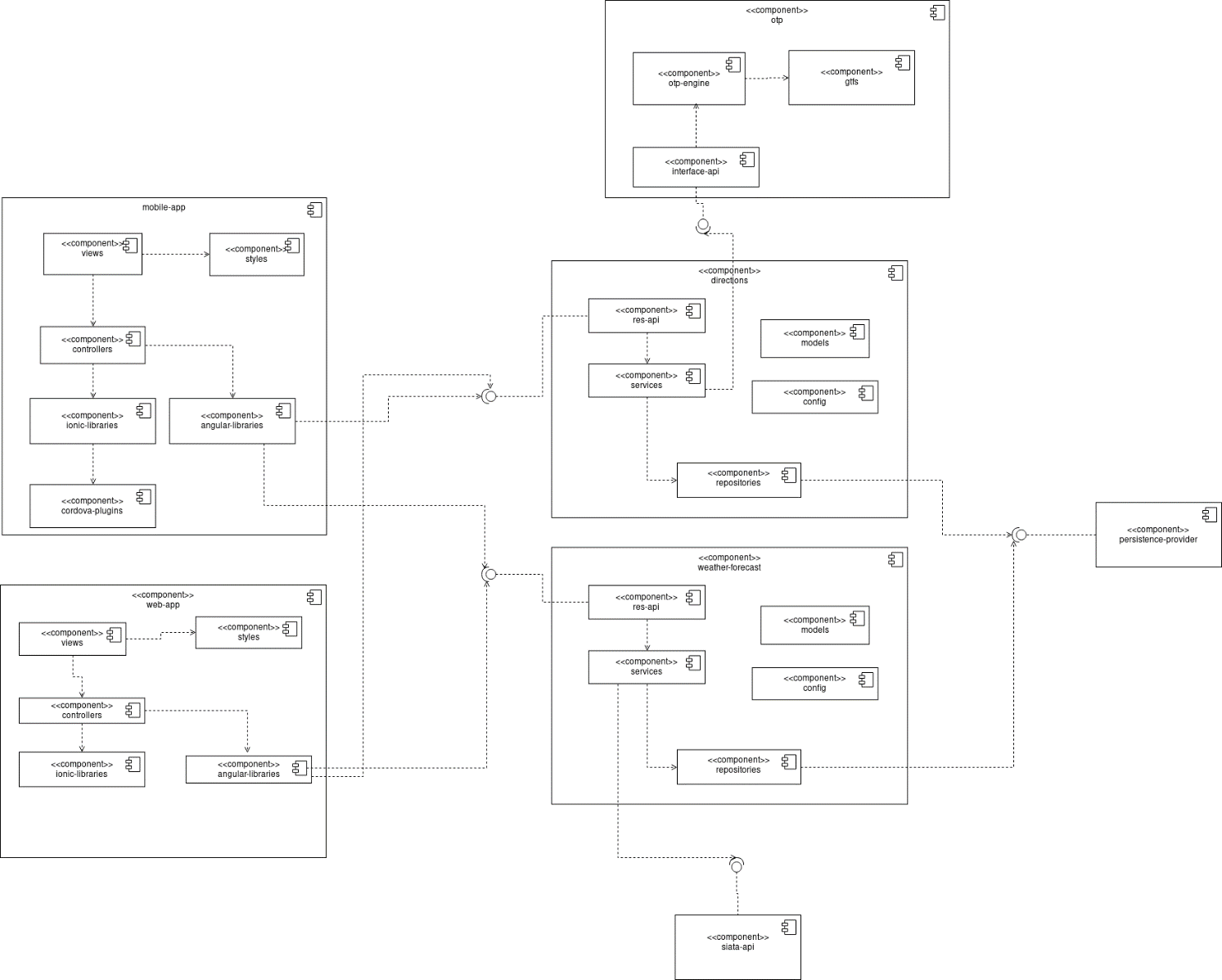
Las compañías o entidades de transporte ingresan información actualizada de sus rutas, esto es, polylines de ruta, incluyendo stop-times, y velocidad comercial de viaje, con el fin de que sean cargadas por el sistema de información, para que este a su vez las incluya en los nuevos cálculos de viajes que realizan los usuarios/pasajeros.

C4 Model: Nivel 1 Contenedores



El sistema Quick Bus se conforma de 2 grandes frentes de trabajo, uno donde interactuarán el ente gubernamental y la compañía de transporte haciendo las respectivas parametrizaciones del sistema vía web accediendo a una aplicación SPA que interactúa mediante el protocolo HTTPS/REST en formato JSON con el API de notificaciones y comunicaciones para acceder a la base de datos que se encuentra en una infraestructura en nube, dichas parametrizaciones van relacionadas con la gestión de rutas y buses y adicionalmente la generación de informes solicitados. El pasajero tendrá acceso al sistema Quick Bus mediante una aplicación móvil desplegada en Ionic/Angular, dicha aplicación proveerá las funcionalidades de ruteo, visualización del pronóstico del clima y alerta de llegada de los buses haciendo uso de los servicios de geolocalización al comunicarse con el API de geolocalización; por otro lado el pasajero también tendrá la opción de registrarse en el sistema y tener un perfil propio, para esto se accederá a la base de datos que se encuentra desplegada en la infraestructura en nube mediante el API de microservicio de notificaciones y comunicaciones, esta comunicación se realiza con protocolo HTTPS/REST en formato JSON.

C4 Model: Nivel 2 Componentes



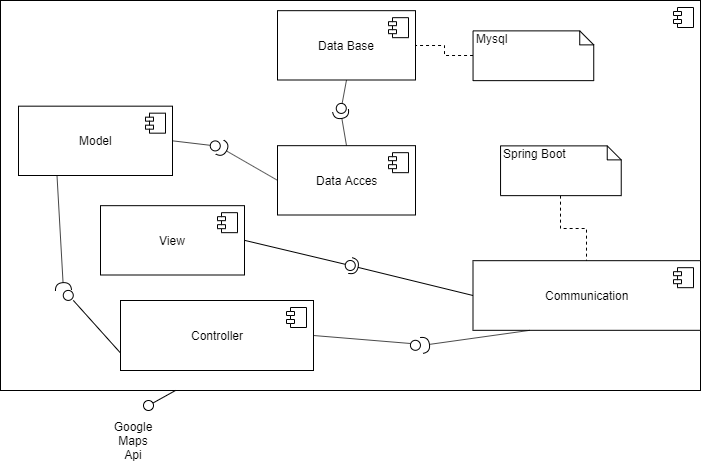
https://drive.google.com/open?id=1i9boK\_RLb8ZSXH2atPlwn2DB\_VvOW1qi

Descripción de los elementos del diagrama de componentes:

|  |  |
| --- | --- |
| **Elemento** | **Descripción** |
| views | Se define todo lo relacionado a la interacción hombre-máquina que tendrá el sistema, mediante el SDK Ionic y el framework angular. |
| styles | Se define toda la hoja en estilo de cascada (CSS), estableciendo el diseño visual de la interfaz. |
| controllers | Componente encargado de solicitar datos y comunicarlo a la vista |
| angular-libraries | La aplicación esta soportada con la librería de Google Maps para la implementación de sus servicios. |
| ionic-libraries | Al ser un proyecto ligado a angular ionic sus librerías están fuertemente ligadas. |
| cordova-plugins | Se instalo el plugin de Google maps para Córdova para las plataformas de Android, IOS y Browser. |
| rest-api | Conjunto de restricciones para el desarrollo de la aplicación mediante el protocolo HTTP. |
| services | Representa los servicios implementados en la aplicación, con sus respectivas operaciones para manipular los recursos. |
| models | Componente encargado de manipular y gestionar los datos según el tipo de operación a realizar. |
| config | Representa la configuración inicial de la aplicación del usuario. |
| repositories | Componente en el cual se almacena la información del proyecto implementado. |
| persistence-provider | Servicio inyectable a un componente del sistema. |
| siata-api | Componente que representa el servicio proporcionado en relación al estado meteorológico en un punto especifico. |
| interface-api | Conjunto de funciones o servicios que ofrece el microservicio para ser utilizado por el usuario. |
| otp-engine | Autenticación de contraseña de un solo uso utilizada por las empresas de transporte. |
| gtfs | Especificación general de los horarios de transporte público y su respectiva información geográfica utilizada por el usuario. |

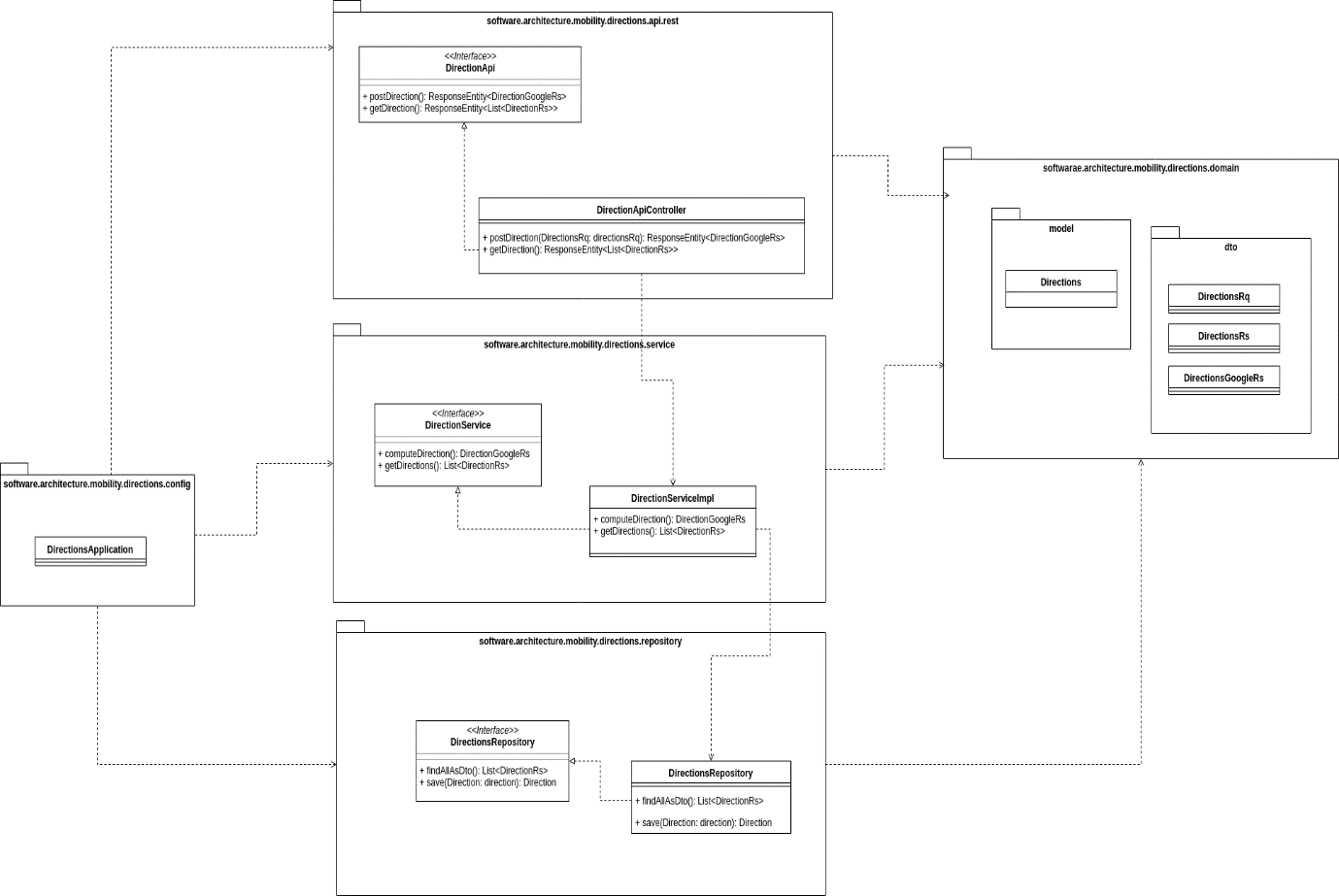
C4 Model: Nivel 3 Código

Diagrama de componentes de la aplicación móvil



Los servicios proporcionados son mediante un sistema de comunicación, esta información es requerida por la interfaz y el controlador de la aplicación para su posterior almacenamiento, la aplicación requiere de una api externa de Google para el manejo de información.

Diagrama de clases/paquetes del microservicio Directions

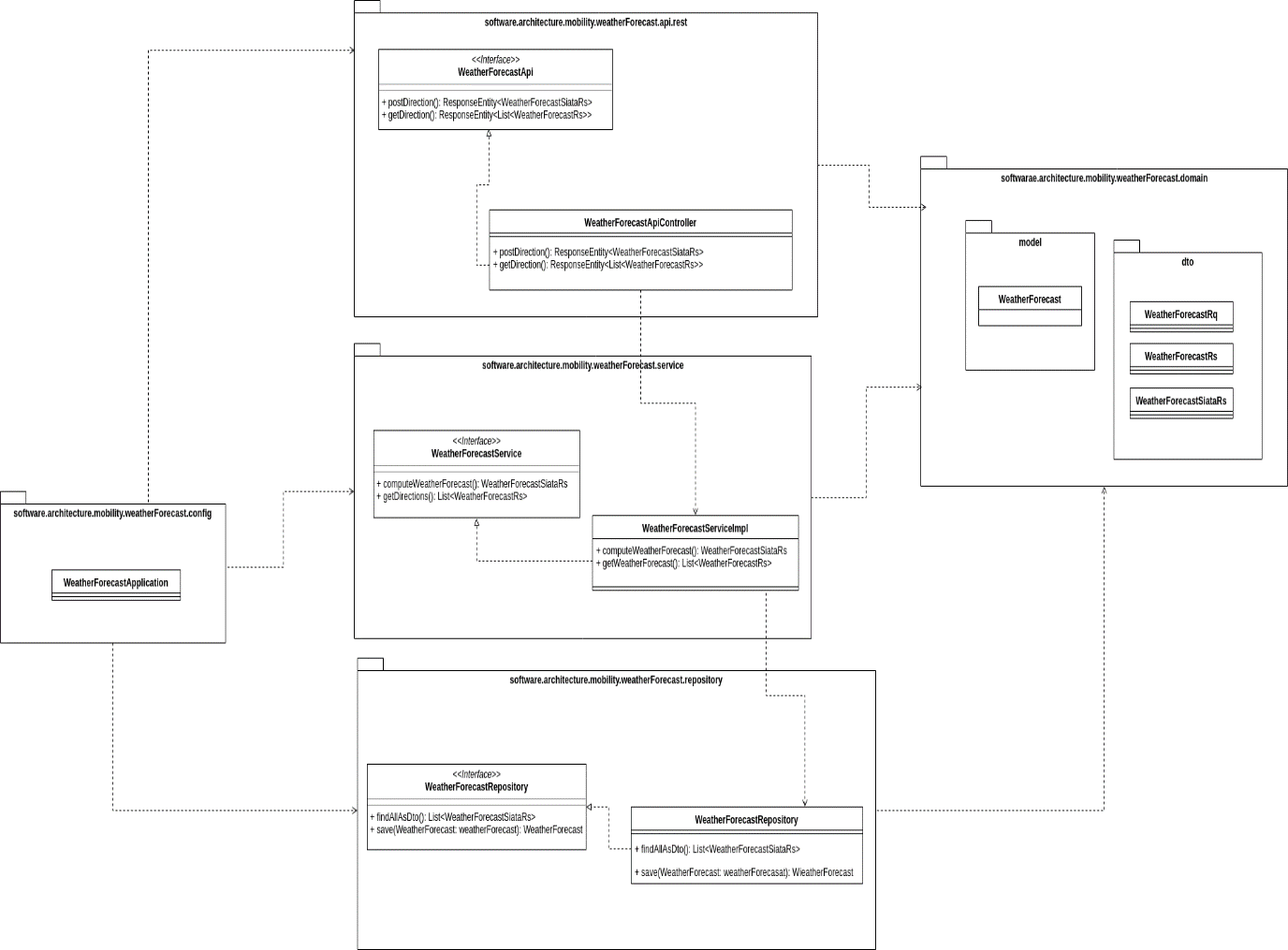


https://drive.google.com/open?id=1cNk2y8mM5T4V2HiSO1n7GOre15kQj91j

Descripción de los paquetes del microservicio Directions:

|  |  |
| --- | --- |
| **Paquete** | **Descripción** |
| software.architecture.microservices.movilidad.directions.api.rest; | Agrupador para la capa de exposición de servicios. Aquí están las interfaces que declaran los contratos de uso del microservicio, esto es, la forma en que se reciben requests y se envian responses, además de los end-points, parámetros, etc para el microservicio.  También se encuentran aquí las clases que implementan estas interfaces. |
| software.architecture.microservices.movilidad. directions.service | Agrupador para la capa de lógica (services) del microservicio. Aquí están declaradas y definidas las capacidades del microservicio. Contiene toda la lógica del microservicio. |
| software.architecture.microservices.movilidad. directions.repository | Agrupador para la capa de acceso a datos del microservicio. Aquí están implementadas las clases que brindan el acceso a datos del microservicio, tanto para obtener información como para persistirla. |
| software.architecture.microservices.movilidad. directions.domain | Agrupador para la capa de modelos. Aquí se definen todas las entidades (clases tipo POJO (Plain Old Java Object)) de negocio que son utilizadas por el microservicio y que tienen mapeo directo con tablas en la base de datos. Aquí también se definen los DTOs (Data Transfer Objects) usados por el microservicio.  Esta capa es transversal a todo el microservicio, lo que quiere decir que cualquier otra capa puede realizar llamados directos a esta. |
| software.architecture.microservices.movilidad. directions.config | Agrupador para las clases de configuración del microservicio. Aquí se definen Java Beans para la configuración de Logger, seguridad del microservicio, configuraciones internas, inicio de ejecución del microservicio, etc. |

Diagrama de clases del microservicio Weather-forecast



https://drive.google.com/open?id=1ereYxLaQeVb2aU1QvHf6JxZ\_y78JFso\_

Descripción de los paquetes del microservicio Weather-forecast:

|  |  |
| --- | --- |
| **Paquete** | **Descripción** |
| software.architecture.microservices.movilidad.weatherForecast.api.rest; | Agrupador para la capa de exposición de servicios. Aquí están las interfaces que declaran los contratos de uso del microservicio, esto es, la forma en que se reciben requests y se envian responses, además de los end-points, parámetros, etc para el microservicio.  También se encuentran aquí las clases que implementan estas interfaces. |
| software.architecture.microservices.movilidad.weatherForecast.service | Agrupador para la capa de lógica (services) del microservicio. Aquí están declaradas y definidas las capacidades del microservicio. Contiene toda la lógica del microservicio. |
| software.architecture.microservices.movilidad.weatherForecast.repository | Agrupador para la capa de acceso a datos del microservicio. Aquí están implementadas las clases que brindan el acceso a datos del microservicio, tanto para obtener información como para persistirla. |
| software.architecture.microservices.movilidad.weatherForecast.domain | Agrupador para la capa de modelos. Aquí se definen todas las entidades (clases tipo POJO (Plain Old Java Object)) de negocio que son utilizadas por el microservicio y que tienen mapeo directo con tablas en la base de datos. Aquí también se definen los DTOs (Data Transfer Objects) usados por el microservicio.  Esta capa es transversal a todo el microservicio, lo que quiere decir que cualquier otra capa puede realizar llamados directos a esta. |
| software.architecture.microservices.movilidad.weatherForecast.config | Agrupador para las clases de configuración del microservicio. Aquí se definen Java Beans para la configuración de Logger, seguridad del microservicio, configuraciones internas, inicio de ejecución del microservicio, etc. |

# Estrategia de solución

La aplicación Quick Bus pretende resolver un problema común y cotidiano de la comunidad de Medellín el cual se centra en la movilidad del municipio, con la aplicación se ofrecen la capacidad de ver las mejores rutas disponibles de acuerdo a la ubicación actual del pasajero y el destino al cual desea llegar.

La aplicación se diseña con base al cumplimiento de los atributos de calidad que se definen como más importantes:

* + Disponibilidad
  + Portabilidad
  + Eficiencia de desempeño
  + Usabilidad
  + Mantenibilidad.

Teniendo en cuenta las decisiones de diseño se realiza el desarrollo del Front-end o interfaces de usuario mediante la tecnología hibrida que ofrece el framework Ionic/Angular, esto impulsa la mantenibilidad, portabilidad y usabilidad del producto; en cuanto al back-end se decide utilizar como plataforma de despliegue Google Cloud Platform haciendo uso de su característica de infraestructura como servicios, logrando así la disponibilidad y eficiencia en desempeño con los que debe cumplir el sistema, dicha estrategia aplicada reduce las responsabilidades del equipo de desarrollo en cuanto a tareas de parametrización y configuración de infraestructura.

Se implementaron 2 Microservicios:

* + Directions, está encargado de proponer la mejor ruta tomando en cuenta el punto de origen (Ubicación actual del pasajero) y el punto destino (estación a la cual se dirige), esto apoyado en el consumo del api de google directions.
  + Weather forecast, es el encargado de entregar un pronóstico del estado del tiempo tanto en el punto de origen como en el punto destino, teniendo en cuenta la hora de llegada estimada; esto es posible al consumir las funcionalidades que provee el api del SIATA.

Los datos provenientes de los viajes realizados son almacenados en una base de datos SQL provista por la infraestructura de Google Cloud Platform, dicha información será vital para la generación y entrega de los informes solicitados por la alcaldía de Medellín y las empresas de transporte.

**Estilos Arquitectónicos Implementados:**

Los estilos arquitectónicos bajo los cuales se implementa el proyecto son:

* **SOA (Service Oriented Architecture):** Para la comunicación entre nodos con roles cliente y servidor. Se define de manera específica unas interfaces de comunicación, también llamadas contratos de servicios, que independientemente de la tecnología utilizada, prescriben la forma en que se comunican clientes y servidor. En el cliente, ya sea móvil o web, se implementa el estilo arquitectónico basado en componentes, es decir, se tiene un conjunto de componentes, cada uno con responsabilidades bien definidas y que a su vez, exponen interfaces bien definidas de modo tal que, a través de componentes controladores, operan colaborativamente para brindar resultados según el proceso definido (codificado).
* **En el servidor, se implementa el estilo arquitectónico por capas así:**
  + **Capa de exposición de servicios:** se definen los contratos de servicios de forma declarativa (abstracción) usando interfaces y también se realiza una implementación de esta abstracción para que pueda ser instanciada.
  + **Capa de servicios:** en esta capa se define toda la lógica de los diferentes procesos implementados.
  + **Capa de acceso a datos:** esta capa habilita la capacidad de acceder a data persistente o persistir data. También se codifica haciendo uso de una abstracción (contrato para persistencia) y una implementación de esta abstracción, al igual que la capa de exposición de servicios.

Transversalmente está la capa de modelos y DTOs (Data Transfer Objects), estos últimos siendo accedidos por las capas anteriormente mencionadas.

* **3-Tier:** En general, el estilo arquitectónico para el despliegue del sistema de infomación es 3-Tier, de la siguiente forma:
  + - Tier (client) encargado de la presentación.
    - Tier (app server) encargado de toda la lógica de negocio.
    - Tier (data server) encargado de la persistencia de data.

**Paradigmas de Programación:**

Los paradigmas de programación utilizados en la implementación del proyecto son:

* + **Paradigma orientado a objetos:** se usa ampliamente en la lógica de aplicación, para la definición de modelos y datos.
  + **Paradigma funcional:** se usa este paradigma en la implementación del cliente y el servidor; los lenguajes de programación Javascript y Java propician el uso de este paradigma brindando soporte a lambdas (funciones anónimas) y Higher-Order functions.

**Patrones de Diseño:**

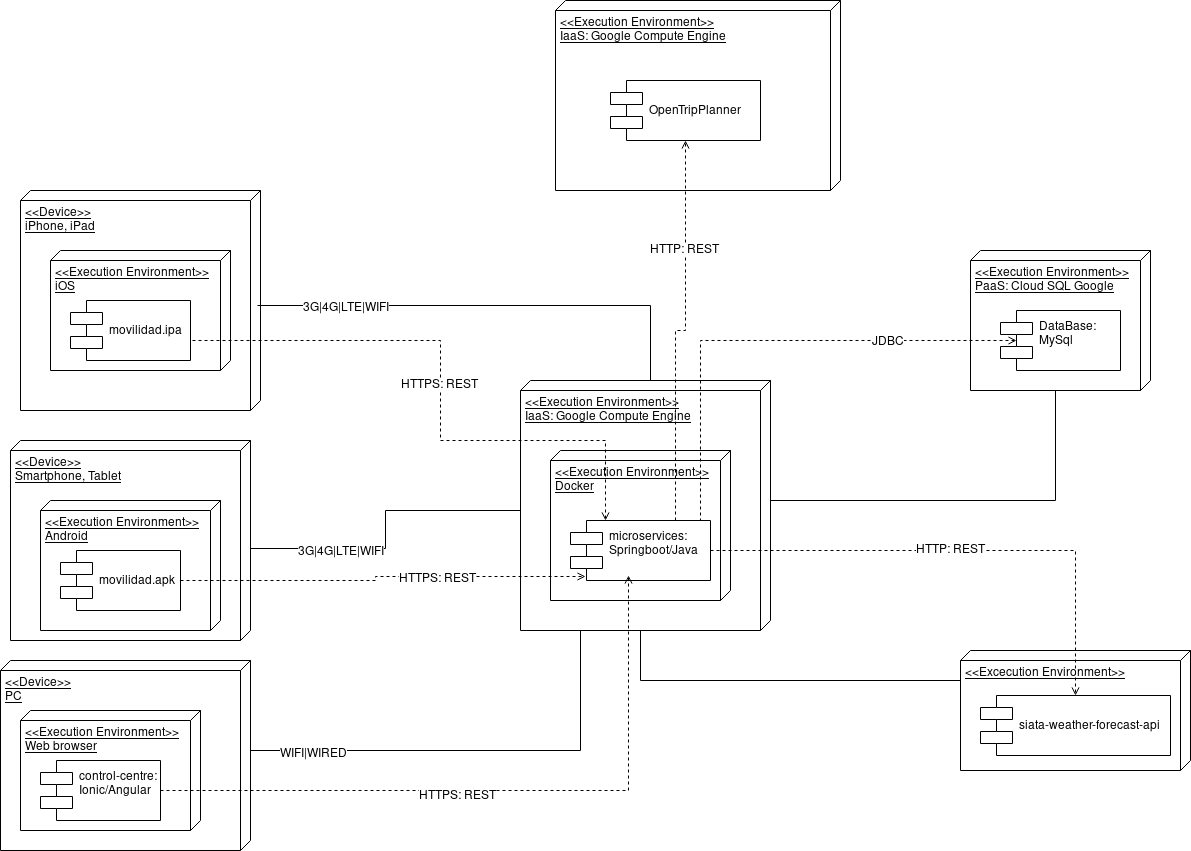
Los patrones de diseño implementados son los siguientes:

* + **DAO (Data Access Object):** Este patrón se usa para acceder a los datos persistentes de modo tal que se ingresa desde una interfaz común, independizando el acceso a datos, del proveedor concreto de esos datos.
  + **Programación asíncrona:** Se hace uso de Callbacks para ejecutar código de manera asíncrona en respuesta a servicios distribuidos que han sido invocados desde el cliente. Esta técnica es común en Javascript, de modo tal que no se afecta la interacción del usuario con el sistema mientras que se realizan peticiones a servidor.
  + **DTO (Data Transfer Object):** Este patrón tiene la finalidad de encapsular un conjunto de datos a transferir (ya sea de cliente a servidor o viceversa), en una clase tipo POJO, de modo tal que se maneje un único objeto, bajo un denominador común (el objeto POJO encapsulador), en vez de un conjunto de parámetros (atributos) no agrupados. Este patrón también permite definir respuestas especificas en servicios expuestos, evitando responder información adicional no requerida desde las entidades de negocio.

**Estándares para Desarrollo:**

Como estándar base para el proceso de codificación, se siguen los principios SOLID, con el objetivo de mejorar la mantenibilidad, legibilidad y la extensibilidad del software en el tiempo. A la par con la aplicación de los principios SOLID al código, se logra ser menos extenso (verboso) en el proceso de documentación del código, pues los bloques de código bien definidos, nombrados, y codificados con intensión, son preferibles a un código poco entendible y extensamente documentado al detalle.

# Vista de despliegue o implementación



<https://drive.google.com/open?id=1NxqNeEo5EFAe1g9RzsvjC78MyxOX2hRB>

Especificación de los nodos del sistema de información

|  |  |  |
| --- | --- | --- |
| **Identificación** | **Nodo** | **Especificación** |
| (1) | <<Device>> iPhone, iPad | Nodo cliente contenedor del tipo: Dispositivo, que categoriza un dispositivo iPhone o iPad |
| (2) | <<Execution Environment>> iOS | Nodo cliente, subnodo de (1), del tipo: Ambiente de ejecución, que categoriza los sistemas operativos de la familia iOS, versiones 9 y superior. |
| (3) | <<Device>> Smartphone, Tablet | Nodo cliente contenedor del tipo: Dispositivo, que categoriza un dispositivo tipo smartphone o tablet diferentes a (1) |
| (4) | <<Execution Environment>> Android | Nodo cliente, subnodo de (3), del tipo: Ambiente de ejecución, que categoriza los sistemas operativos de la familia Android, versiones 5 y superior. |
| (5) | <<Device>> PC | Nodo cliente contenedor del tipo: Dispositivo, que categoriza un dispositivo tipo desktop o laptop. |
| (6) | <<Execution Environment>> Web Browser | Nodo cliente, subnodo de (5), del tipo: Ambiente de ejecución, que categoriza un web browser genérico. |
| (7) | <<Execution Environment>> IaaS Google Compute Engine | Nodo servidor, del tipo: Ambiente de ejecución. Se trata de los servicios de Infrastructure as a Service que provee Google Cloud para la creación de máquinas virtuales en nube pública. |
| (8) | <<Execution Environment>> Docker | Nodo servidor de aplicaciones, subnodo de (7), del tipo: Ambiente de ejecución. Se trata de la instancia de Docker que corre sobre el ambiente de ejecución configurado dentro del IaaS de Google Cloud. |
| (9) | <<Execution Environment>> PaaS Cloud SQL Google | Nodo servidor de datos, del tipo: Ambiente de ejecución. Se trata de los servicios de Platform as a Service que provee Google Cloud para bases de datos SQL. |
| (10) | <<Execution Environment>> | Nodo que representa los servicios distribuidos que son consumidos por nuestro sistema de información. Se desconoce la implementación interna de estas APIs expuestas. Solo se conoce y nos interesa las interfaces de consumo expuestas. |

Especificación de los artefactos del sistema de información

|  |  |  |
| --- | --- | --- |
| **Identificación** | **Artefacto** | **Especificación** |
| (11) | Movilidad.ipa | Representa un archivo con extensión .IPA, el cual contiene todo el software ejecutable para operar en la plataforma representada por el nodo (2). Se trata del software ejecutable como una aplicación móvil iOS.  Se desarrolla usando las tecnologías Ionic 4 /Angular 8, de modo tal que es una aplicación híbrida que luego se compila a la plataforma iOS usando Apache Cordova para esto. |
| (12) | Movilidad.apk | Representa un archivo con extensión .APK, el cual contiene todo el software ejecutable para operar en la plataforma representada por el nodo (4). Se trata del software ejecutable como una aplicación móvil Android.  Se desarrolla usando las tecnologías Ionic 4 /Angular 8, de modo tal que es una aplicación híbrida que luego se compila a la plataforma Android usando Apache Cordova para esto. |
| (13) | Control-centre | Representa el paquete de software de la aplicación web que se ejecuta sobre el nodo (6). Se desarrolla usando las tecnologías Ionic 4/Angular 8. |
| (14) | microservices | Representa un contenedor tipo docker que se ejecuta sobre el nodo (8).  Se trata de una aplicación del tipo SpringBoot - Java 8, con un servidor apache tomcat embebido, que se ha containerizado usando docker. |
| (15) | database | Representa una instancia de base de datos MySql que corre sobre el nodo (9). |
| (16) | Siata-weather-forecast-api | Representa el api de servicios distribuidos del SIATA (Sistema de Alertas Tempranas del Valle de Aburrá) que corre sobre el nodo (10). |

# Conceptos transversales (Cross-cutting)

# Modelos de Dominio

A continuación, se realiza una descripción de las entidades/clases de dominio relacionadas al sistema de información:

|  |  |
| --- | --- |
| **Nombre** | **Descripción** |
| Directions | Entidad encargada de representar un objeto que contiene punto de partida, punto de llegada y un arreglo de opciones de rutas para ir desde el punto de partida al punto de llegada, junto con una caracterización de cada opción de ruta. |
| DirectionsRq | Clase encargada de agrupar todos los parámetros que relacionados a una petición al microservicio Directions |
| DirectionsRs | Clase encargada de representar una respuesta del microservicio Directions |
| OtpRq | Clase encargada de agrupar todos los parámetros que relacionados a una petición al api expuesta por el servidor OTP |
| OtpRs | Clase encargada de representar una respuesta del api expuesta por el servidor OTP |
| SiataRq | Clase encargada de agrupar todos los parámetros que relacionados a una petición al api expuesta por el SIATA |
| SiataRs | Clase encargada de representar una respuesta del api expuesta por el SIATA |
| DirectionsDao | Clase responsable de proveer métodos para el acceso a datos persistentes concernientes a la entidad Directions |
| WeatherForecastDao | Clase responsable de proveer métodos para el acceso a datos persistentes concernientes a la entidad WeatherForecast |
| WeatherForecastService | Clase responsable de la lógica del microservicio, aquí se definen métodos de operacionalizan las transacciones relacionadas a las capacidades definidas para el microservicio. |
| DirectionsService | Clase responsable de la lógica del microservicio, aquí se definen métodos de operacionalizan las transacciones relacionadas a las capacidades definidas para el microservicio. |
| WeatherForecast | Entidad encargada de representar un objeto que contiene punto de partida, punto de llegada y un arreglo de opciones de rutas para ir desde el punto de partida al punto de llegada, junto con una caracterización de cada opción de ruta. |
| WeatherForecastRq | Entidad encargada de agrupar todos los parámetros que relacionados a una petición al microservicio WeatherForecast |
| WeatherForecastRs | Entidad encargada de representar una respuesta del microservicio WeatherForecast |

# Persistencia

Para el ambiente de desarrollo y pruebas, por conveniencia y rápidez de implementación se hace uso de una base de datos H2 in-memory; sin embargo, en la versión final para producción se hace uso de una base de datos relacional MySql.

En cuanto a la forma de acceso a los datos persistentes, se implementa la técnica de programación ORM (Object Relational Mapping) que nos permite abstraer la implementación del modelo de datos y en vez de esto, se trabaja directamente sobre el modelo de objetos, que luego se mapea a modelo de datos de forma automatizada. Para la implementación de esta técnica se utilizan las librerías de JPA (Java Persistence Api).

# Interfaz de Usuario

La interfaz utilizada para el desarrollo de la aplicación está asentada en una aplicación de una sola página, integrando tecnológicas como ionic y angular además de un fuerte ligamiento a librerías para la implementación de mapas que nos ofrece Google.

# Comunicaciones e Integración

La comunicación en el sistema de información desde los nodos clientes a los nodos servidor se realiza a través de web services (microservices) con el protocolo de comunicación REST, usando el formato de transferencia de información JSON; esto es a nivel de estilo arquitectónico, una arquitectura orientada a servicios SOA.

# Excepción y Manejo de Errores

El manejo de errores y excepciones en la api de microservicios es tenido en cuenta, debido a que se acceden a otras Apis de terceros, se consideran las excepciones que se puedan generar en estos request y para toda petición de la api de microservicios se manejan dos parámetros para comunicar el código de estado, es decir, si la petición fue recibida aceptada y atendida (OK), pero si la petición no puede procesarse se debe retornar un parámetro adicional llamado message ,proporcionando una descripción human-readable del error ocurrido. Adicionalmente, para cada petición se manejan los códigos de estado de HTTP.

# Logging, Tracing

Con respecto a los microservicios, al ser cada uno una aplicación del tipo Springboot app, el framework se encarga de transmitir logs por default al stream stdout (Standar Output), mediante el registro de Apache Commons, por lo que es una dependencia única y obligatoria. Estos logs por default que trae consigo el framework, se robustecen con la generación de logs explicita dentro del código en operaciones que lo ameriten como por ejemplo el inicio de un request a un servicio distribuido, o en respuesta a la generación de algún tipo de excepción, por defecto los errores se pueden encontrar en 3 lineas:Error, Warn e Info.

Para el ambiente de desarrollo, la configuración de nivel de log se setea en el nivel DEBUG, de modo tal que se obtenga gran cantidad de información de logs y stack trace (trazabilidad en pila de llamadas de métodos) de los mismos.

# Configurabilidad

Siendo nuestro caso él de una aplicación de Springboot, tenemos un conjunto de configuraciones por default que a su vez son transversales a todos los perfiles de configuración que se definan. Este archivo de configuración se encuentra en la ruta: {raiz-del-proyecto} src/main/resources/application.properties. En este archivo se encuentran las siguientes configuraciones:

|  |  |
| --- | --- |
| **Configuración** | **Descripción** |
| Directions.server.port = 8080 | Puerto por el cual se le da salida a la API REST del microservicio Directions |
| WeatherForecast.server.port = 8081 | Puerto por el cual se le da salida a la API REST del microservicio WeatherForecast |

En este archivo, se encuentran todas las configuraciones de cada microservicio, desde puntos de acceso a servidores externos de datos o APIs externas, hasta urls base de los microservicios expuestos, así como política para la ejecución de DDL de JPA.

# Testability

En cuanto a la aplicación móvil y la aplicación web, se utiliza el framework Jasmine para la construcción de pruebas unitarias y se utiliza la herramienta Karma para la ejecución de estas.

En cuanto a los microservicios, se utiliza JUnit como framework para pruebas.

# Internationalizacion

Dado que la aplicabilidad inicial del proyecto, definida en el alcance, contempla el despliegue del sistema de información en alguna localidad (municipio) del Valle de Aburrá, el idioma principal soportado es el Español-Colombia. A futuro, se considerará la inclusión del idioma Inglés, dada la afinidad que pudiesen tener los usuarios extranjeros con el uso de este sistema de información para facilitar sus desplazamientos dentro del territorio, utilizando medios de transporte público.

# Build-Management

En cuanto a la aplicación móvil y la aplicación web, dado el framework que se utiliza (Ionic 4), la herramienta que automatiza la construcción del proyecto es Gradle.

En cuanto a los microservicios, se utiliza Maven para la construcción del software ejecutable con sus dependencias.

Adicionalmente, se usa Docker para la construcción de microservicios como containers (containerización).

# Decisiones de diseño

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Decisión** | **Mantenibilidad** | **Desempeño** | **Portabilidad** | **Disponibilidad** | **Usabilidad** |
| Uso de tecnologías nativas | Se ve afectada negativamente en un alto porcentaje ya que al momento de realizar ajustes o correcciones se deben realizar para cada plataforma en la que se haya desplegado la aplicación. | Se afecta positivamente puesto que el lenguaje nativo aprovecha completamente las características del hardware. | Al desarrollar la app en lenguaje nativo sólo se podrá acceder a un único sistema operativo, por lo tanto, la portabilidad se reduce sólo a una única plataforma. | La elección de la plataforma de desarrollo no afecta este atributo. | Puede ofrecerse una buena experiencia de usuario, pero el esfuerzo del equipo de desarrollo se verá incrementado. |
| Uso de tecnologías hibridas | Este atributo se ve impulsado positivamente porque los ajustes y modificaciones a la aplicación móvil se realizan directamente desde una única instancia de desarrollo y se refleja en diversos sistemas operativos. | El desempeño puede verse levemente afectado porque hay mayor esfuerzo en la interacción del software con el hardware. | Al desarrollar la app con tecnologías hibridas se hace más fácil escalarla a otros sistemas operativos. | La elección de la plataforma de desarrollo no afecta este atributo. | Aumenta porque se transmite una buena experiencia de usuario con menos esfuerzo por parte del equipo de desarrollo |
| Replicar suite de servicios | Presenta ciertas dificultades en el proceso de implementacióny despliegue porque se hace necesario mantener y administrar las “n” instancias que se repliquen. | Con una correcta implementación se puede lograr un escalamiento horizontal cuando el número de usuarios concurrentes sea más elevado. | La decisión de esta implementación no afecta este atributo. | Aumenta considerablemente porque de implementarse bien se tiene una contingencia en caso de fallos en una de las instancias de los servicios. | La decisión de esta implementación no afecta este atributo. |
| Diseño responsive | Se incrementa considerablemente porque se reduce la deuda técnica en cuanto al desarrollo de la capacidad de la aplicación en ser compatible en varios dispositivos y configuraciones. | La decisión de esta implementación no afecta este atributo. | La portabilidad aumenta considerablemente porque la aplicación puede funcionar correctamente en diversas configuraciones de hardware en los variados dispositivos. | La decisión de esta implementación no afecta este atributo. | Mejora ostensiblemente la experiencia de usuario, al ofrecerle un diseño congruente independientemente del dispositivo que se utilice. |
| Uso de microservicios | La mantenibilidad aumentaría en caso de definir de forma correcta las funcionalidades | Al tener implementados microservicios con una única funcionalidad, el desempeño se vería afectado positivamente porque cada módulo de código puede dedicarse a una actividad específica. | La decisión de esta implementación no afecta este atributo. | Al tener funcionalidades únicas se facilita la detección y corrección de fallos, y en el caso dado permiten retornar la disponibilidad de manera más oportuna. | La decisión de esta implementación no afecta este atributo. |
| Google cloud: Infrastructure as a service. | Mejora porque el equipo de desarrollo no tiene que ocuparse de los detalles propios de la infraestructura. | Aumenta al ofrecer elasticidad a las capacidades requeridas desde la aplicación. | La decisión de esta implementación no afecta este atributo. | Aumenta considerablemente ya que este tipo de plataformas ofrecen alta disponibilidad. | La decisión de esta implementación no afecta este atributo. |

Después de analizar como las diferentes opciones de implementación afectan los atributos de calidad que definimos como importantes para el proyecto se llega a la decisión de implementar tecnologías hibridas (Ionic) para el desarrollo de la aplicación móvil y Google Cloud Infrastructure AAS para la implementación funcional en backend.

# Escenarios de calidad

|  |  |
| --- | --- |
| **Fuente del evento** | Pasajero |
| **Estimulo** | Acceder a la opción de consultar la ruta existente entre puntos A y B. |
| **Ambiente** | Quick Bus |
| **Artefacto** | Módulo consulta de rutas |
| **Respuesta** | Representación gráfica de la ruta entre los puntos A y B en el mapa de la ciudad. |
| **Medición de la respuesta** | El usuario no visita más de 2 pantallas para acceder a la información. |

|  |  |
| --- | --- |
| **Fuente del evento** | Pasajero |
| **Estimulo** | Acceso a la ubicación del usuario en tiempo real mediante el uso del GPS del celular |
| **Ambiente** | Quick Bus |
| **Artefacto** | Módulo ubicación del usuario |
| **Respuesta** | Ubicación del usuario en tiempo real dentro del mapa de la ciudad |
| **Medición de la respuesta** | El sistema identifica la ubicación del usuario en tiempo real en el mapa de la ciudad mediante el uso del GPS de forma rápida. |

|  |  |
| --- | --- |
| **Fuente del evento** | Pasajero |
| **Estimulo** | El usuario desea instalar la aplicación Quick Bus en un dispositivo Android y en un iPhone. |
| **Ambiente** | Sistema operativo de cada dispositivo |
| **Artefacto** | Archivo instalador de aplicación Quick Bus para dispositivos Android y iOS. |
| **Respuesta** | Según el archivo de instalación utilizado para la correspondiente plataforma se instalará la aplicación. |
| **Medición de la respuesta** | Mensaje de instalación exitosa en cualquiera de las 2 plataformas. |

|  |  |
| --- | --- |
| **Fuente del evento** | Software de prueba JMeter |
| **Estimulo** | Simulación de conexión de 10.000 usuarios concurrentes |
| **Ambiente** | Quick Bus |
| **Artefacto** | Módulo consulta de rutas |
| **Respuesta** | Representación gráfica de la ruta establecida entre los puntos A y B en el mapa de la ciudad para cada uno de los usuarios conectados. |
| **Medición de la respuesta** | La representación gráfica de la ruta debe darse en un lapso no mayor a 10 segundos. |

|  |  |
| --- | --- |
| **Fuente del evento** | Manualmente desactivar una de las instancias que proveen el microservicio de login. |
| **Estimulo** | Simulación de fallo de microservicio de login |
| **Ambiente** | Google cloud platform |
| **Artefacto** | Módulo de login |
| **Respuesta** | La plataforma restablecerá el servicio haciendo uso de la réplica del microservicio. |
| **Medición de la respuesta** | La aplicación debe estar disponible pasado 1 minuto. |

|  |  |
| --- | --- |
| **Fuente del evento** | Pasajero |
| **Estimulo** | Intentar acceder a un perfil diferente al propio |
| **Ambiente** | Quick Bus |
| **Artefacto** | Módulo login |
| **Respuesta** | La aplicación denegará el acceso al usuario y presentará mensaje de credenciales erróneas. |
| **Medición de la respuesta** | El mensaje de credenciales erróneas será retornado inmediatamente. |

|  |  |
| --- | --- |
| **Fuente del evento** | SonarQube |
| **Estimulo** | Verificar líneas de código duplicado en el desarrollo de la aplicación. |
| **Ambiente** | Quick Bus |
| **Artefacto** | Aplicación Quick Bus en fase de desarrollo |
| **Respuesta** | Obtener número de líneas duplicadas en los archivos del código fuente. |
| **Medición de la respuesta** | El test de SonarQube debe arrojar un porcentaje de líneas de código duplicadas inferior al 10%. |

# Riesgos y deuda técnica

No se consideran riesgos importantes en cuanto al desarrollo y utilización de la aplicación Quick Bus dado que no se transa ni se comparte información sensible de los pasajeros ni de las empresas transportadoras.

En cuanto a posibles fallas de la infraestructura se delegaron las responsabilidades a los proveedores de plataformas en la nube, en caso de una falla en la conexión a internet la aplicación presentará el correspondiente mensaje de error y estará indisponible.

# Glosario

**Android:** Sistema operativo móvil desarrollado por Google, basado en el Kernel de Linux y otros software de código abierto.

**API:** Interfaz de programación de aplicaciones, en inglés *Application Programming Interface.* Hace referencia a los procesos, funciones y métodos que brinda una determinada biblioteca de programación a modo de capa de abstracción para que sea empleada por otro programa.

**Back-end:** Parte del programa que se encuentra en servidor, lugar donde se despliega el código necesario para cumplir con las funcionalidades de negocio.

**Front-end:** Parte del programa que se encarga de la interacción con el usuario, normalmente desplegado en entornos web o aplicaciones móviles.

**Geolocalización:** Obtener la localización de un punto geográfico exacto que es determinado por medio de ciertas coordenadas obtenidas de información emitida por satélites.

**Google Cloud Platform:** Plataforma que ha reunido todas las aplicaciones de desarrollo web, es utilizada para crear ciertos tipos de soluciones a través de la tecnología almacenada en la nube y permite por ejemplo destacar la rapidez y la escalabilidad de su infraestructura en las aplicaciones.

**GPS:** Sistema de Posicionamiento Global (GPS por su sigla en inglés: Global Positioning System) es un sistema de navegación basado en satélites y está integrado por 24 satélites puestos en órbita por el Departamento de defensa de los Estados Unidos.

**Infrastructure as a service:** (IaaS) se refiere a los servicios on-line que proporcionan un alto-nivel de APIs utilizadas para indirecciónar detalles a bajo nivel de infraestructura como recursos de informática física, ubicación, dato partitioning, scaling, seguridad, copia de seguridad etc.

**Ionic Framework:** Kit de herramientas de interfaz de usuario móvil gratuito y de código abierto para desarrollar aplicaciones multiplataforma de alta calidad para iOS, Android y la Web nativos, todo desde una única base de código.

**iOS:** Sistema operativo móvil de la multinacional Apple Inc. Originalmente desarrollado para el iPhone, después se ha usado en dispositivos como el iPod touch y el iPad. No permite la instalación de iOS en hardware de terceros.

**Java beans:** Modelo de componentes creado por Sun Microsystems para la construcción de aplicaciones en Java. Se usan para encapsular varios objetos en un único objeto, para hacer uso de un solo objeto en lugar de varios más simples.

**Microservicios:** Representan un estilo de arquitectura y un modo de programar software. Con los microservicios, las aplicaciones se dividen en sus componentes más pequeños, y son independientes entre sí.

**SIATA:** Sitio web del municipio de Medellín que presta funcionalidades geográficas sobre el Valle de Aburrá, sigla que nace del nombre:Sistema de Alerta Temprana de Medellín y el Valle de Aburrá.

**Stakeholders:** Involucrados, parte interesada o interesados hace referencia a una persona, organización o empresa que tiene interés en el desarrollo del proyecto desde la parte del negocio.

**OTP:** Open Trip Planner, es una familia de proyectos de software de código abierto que brindan información de pasajeros y servicios de análisis de redes de transporte. El componente Java central del lado del servidor encuentra itinerarios que combinan segmentos de tránsito, peatones, bicicletas y automóviles a través de redes construidas a partir de datos OpenStreetMap y GTFS de estándar abierto y ampliamente disponibles. Se puede acceder a este servicio directamente a través de su API web o utilizando una variedad de bibliotecas cliente de Javascript, incluidos los componentes modulares reactivos modernos destinados a plataformas móviles.

**ORM:** (Object Relational Mapping), es una técnica de programación para convertir datos entre el [sistema de tipos](https://es.wikipedia.org/wiki/Sistema_de_tipos) utilizado en un [lenguaje de programación](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n) [orientado a objetos](https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos) y la utilización de una [base de datos relacional](https://es.wikipedia.org/wiki/Base_de_datos) como [motor de persistencia](https://es.wikipedia.org/wiki/Motor_de_persistencia). En la práctica esto crea una [base de datos orientada a objetos](https://es.wikipedia.org/wiki/Base_de_datos_orientada_a_objetos) virtual, sobre la base de datos relacional. Esto posibilita el uso de las características propias de la orientación a objetos (básicamente [herencia](https://es.wikipedia.org/wiki/Herencia_(inform%C3%A1tica)) y [polimorfismo](https://es.wikipedia.org/wiki/Polimorfismo_(inform%C3%A1tica))).

**JPA:** (Java Persistence Api), es la [API](https://es.wikipedia.org/wiki/API) de [persistencia](https://es.wikipedia.org/wiki/Persistencia_(inform%C3%A1tica)) desarrollada para la [plataforma](https://es.wikipedia.org/wiki/Plataforma_(inform%C3%A1tica)) [Java EE](https://es.wikipedia.org/wiki/Java_EE). Es un [framework](https://es.wikipedia.org/wiki/Framework) del [lenguaje de programación](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n) [Java](https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)) que maneja [datos relacionales](https://es.wikipedia.org/wiki/Base_de_datos_relacional) en aplicaciones usando la Plataforma Java

**REST:** Representational State Transfer, es un estilo de arquitectura software para sistemas [hipermedia](https://es.wikipedia.org/wiki/Hipermedia) distribuidos como la [World Wide Web](https://es.wikipedia.org/wiki/World_Wide_Web)

**JSON:** Es un formato de texto sencillo para el [intercambio de datos](https://es.wikipedia.org/w/index.php?title=Intercambio_de_datos&action=edit&redlink=1).

**SOA:** Arquitectura orientada a servicios, es un estilo de arquitectura de TI que se apoya en la orientación a servicios.

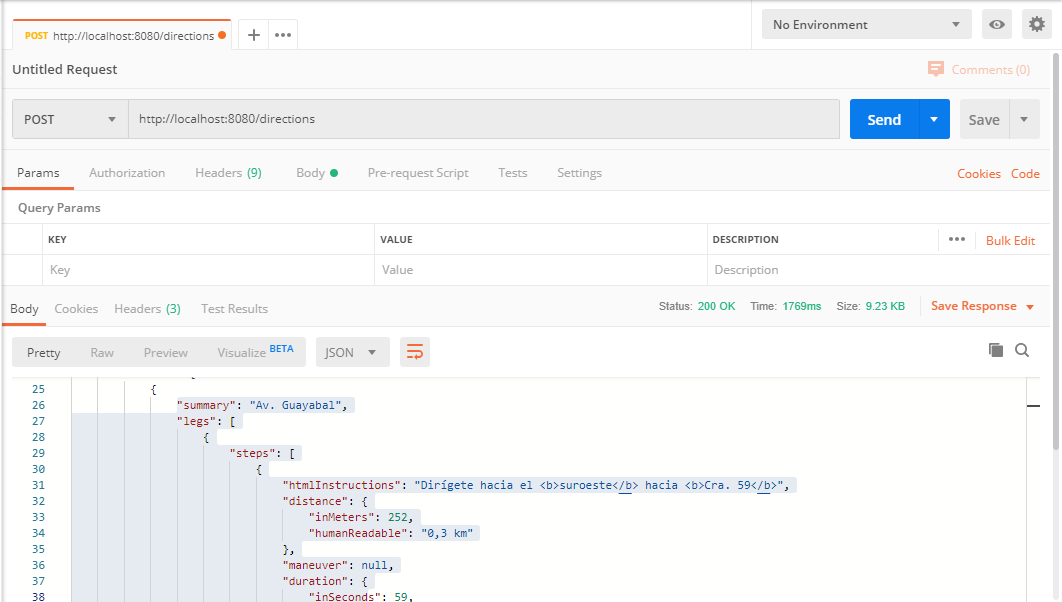
**HTTP:** Hypertext Transfer Protocol, es el [protocolo](https://es.wikipedia.org/wiki/Protocolo_de_comunicaciones) de comunicación que permite las transferencias de información en la [World Wide Web](https://es.wikipedia.org/wiki/World_Wide_Web).

**DDL:** Data Definition Language, Es un lenguaje de programación para definir estructuras de datos, proporcionado por los sistemas gestores de bases de datos,

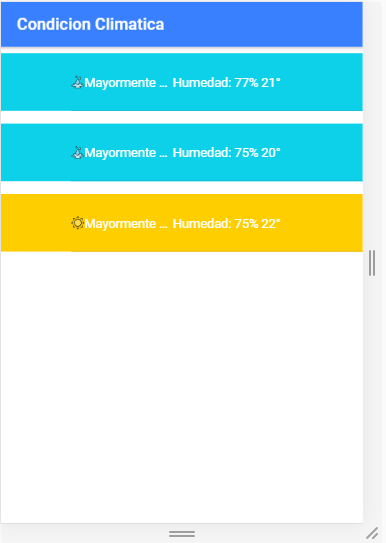
**Jasmine:** Framework para realizar testing de código javascript.

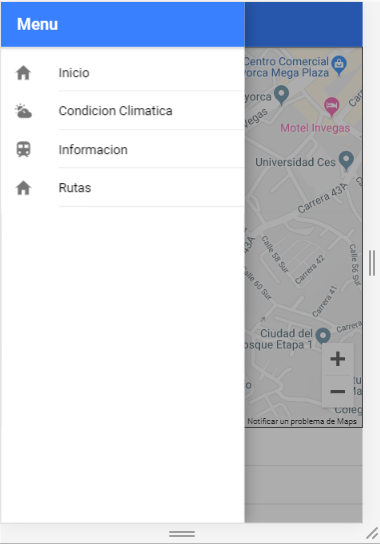
**Karma:** Herramienta de ejecución de código javascript en múltiples browsers. Hace el papel de ejecutor de tests construidos en Jasmine y brinda un dashboard de ejecución.

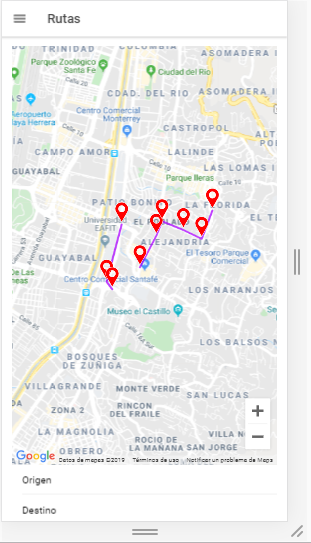
# Anexos











# Bibliografía

* + <https://www.opentripplanner.org/>
  + <https://siata.gov.co/siata_nuevo/>
  + <https://ionicframework.com/>
  + <https://www.redhat.com/es/topics/microservices>
  + https://es.wikipedia.org/wiki/Mapeo\_objeto-relacional
  + <https://es.wikipedia.org/wiki/Java_Persistence_API>
  + <https://iso25000.com/index.php/normas-iso-25000/iso-25010>
  + <https://es.wikipedia.org/wiki/Transferencia_de_Estado_Representacional>
  + <https://es.wikipedia.org/wiki/JSON>
  + <https://es.wikipedia.org/wiki/Protocolo_de_transferencia_de_hipertexto>
  + <https://todopostgresql.com/diferencias-entre-ddl-dml-y-dcl/>